

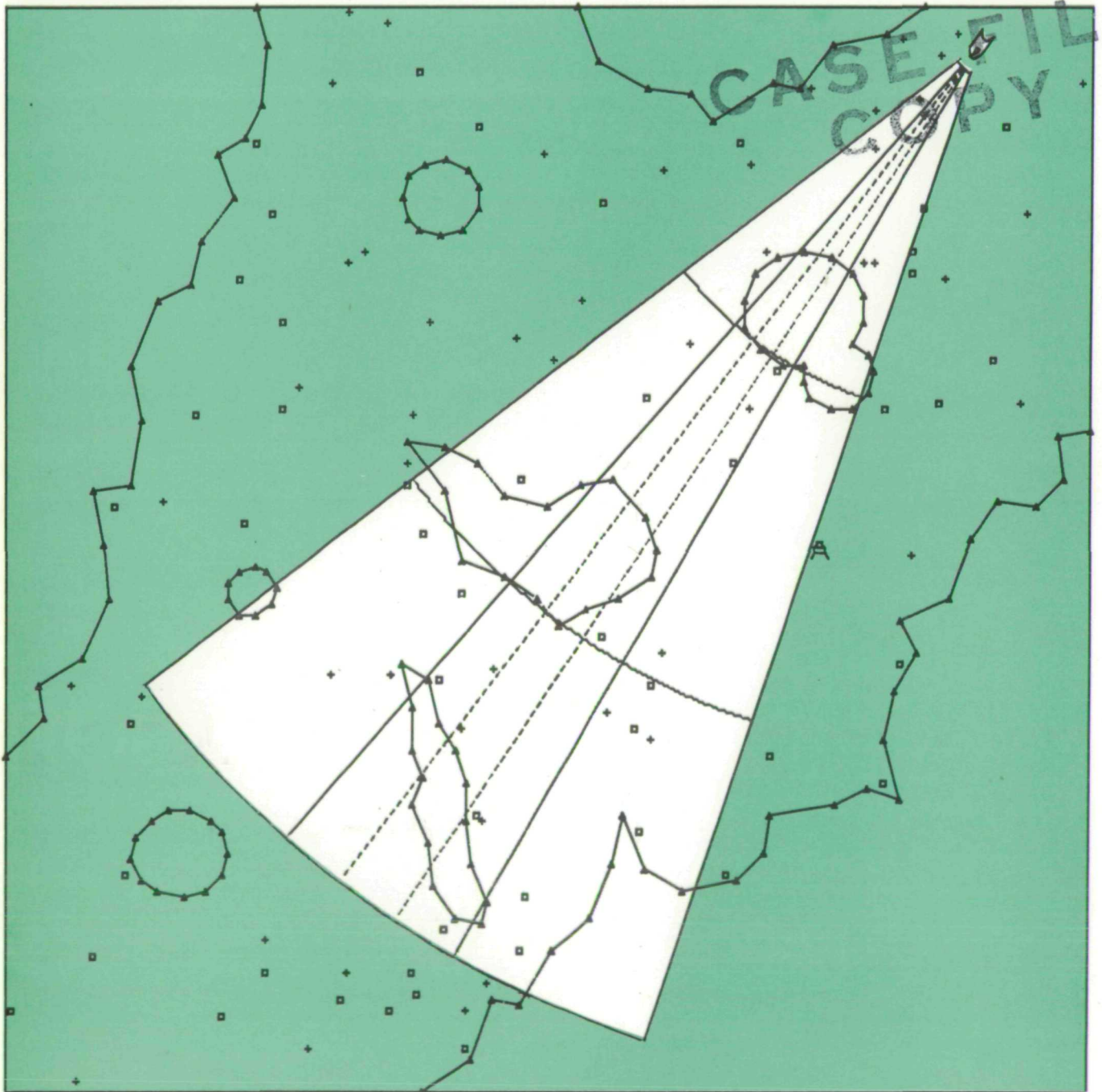
# ROBOT

## Computer Problem Solving System

Bolt Beranek and Newman Inc.  
Report No. 2646

September 1973

Joseph D. Becker  
E. William Merriam



prepared for  
National Aeronautics and Space Administration  
Washington, D.C. 20546

BBN REPORT No. 2646

"ROBOT" COMPUTER PROBLEM SOLVING SYSTEM

FINAL PROGRESS REPORT

CONTRACT No. NASW-2236

JOSEPH D. BECKER  
E. WILLIAM MERRIAM

30 SEPTEMBER 1973

PREPARED FOR:

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION  
WASHINGTON, D.C. 20546

## TABLE OF CONTENTS

	<u>page</u>
I. <u>Introduction</u> .....	1
II. <u>The "Robot" Computer Problem-Solving System</u> .....	4
A. The Robot and its world .....	4
B. The Visual System .....	9
B.1. Point Objects and Features .....	9
B.2. Vision .....	13
B.3. Visual Occlusion .....	17
B.4. Object Reconstruction .....	21
C. Tracking .....	27
D. Robot Display .....	41
III. <u>Assistance with JPL Robot Project</u> .....	44
A. Initial Planning Efforts .....	45
B. Conversion of SAIL to TENEX .....	46
B.1. Compatability Mode SAIL .....	46
B.2. Establishment of TENEX/10-50 Master Files .....	48
B.3. A New IO Package .....	49
B.4. Additions and Changes .....	50
B.5. Testing and Benchmarks .....	51
B.6. LEAP .....	51
C. Graphics System .....	53
D. Consultation on the ARPANET Technology .....	55
E. Consultation on Artificial Intelligence Research .....	57

Appendix A

Appendix B

## I. Introduction

The following is a final report on progress made on NASA Contract No. NASW-2236. This contract concerns research, development, and consulting in the areas of:

- A. Research and development on a "robot" computer problem-solving system, and
- B. Assistance with the NASA-JPL robot project.

Our research into computer problem-solving systems has revolved around a complex simulated "robot" in a rich simulated-world environment. This simulation is not an end in itself, but rather it is a research instrument that allows us to explore theoretical questions involved in the computer control of robots, without our duplicating the work that is being done at the Jet Propulsion Laboratory, the Stanford Research Institute, and other laboratories in the development of robot hardware. However, since our research during the past year has in fact been largely devoted to the development of the simulation system, we will confine ourselves in Section II of this report to discussing the current state of the robot and its simulated environment. The more theoretical aspects of our research were presented at great length in our previous summary report to NASA (Bolt Beranek and Newman Report

No. 2316, of September 1972 - Sections V and VI) and we will not repeat them here. An explanation of our research methodology is included as Appendix A of the present report. When the reader is unsure why we have done things one way and not another, he may find some clue to our motivations in this Appendix.

Briefly, by the period ending in September 1972 we had developed a robot model which exhibited some interesting visual behavior in the context of a simulated city-street environment. During the year ending September 1973 we have converted this simulation to a new environment representing a landscape such as that of Mars. While retaining almost all of the robot's previous behavioral characteristics, we have markedly enriched its world of possible experiences. At the same time, we have reprogrammed our simulation so as to obtain greatly increased computational speed and efficiency. Finally, we have implemented an interactive graphical display of the robot and its world, which is invaluable in allowing us to follow where the robot goes and what it sees, and to evaluate its problem-solving behavior while this activity is actually going on.

Our basic research in robot problem solving is of long-term interest to NASA and to the NASA-supported robot project at the Jet Propulsion Laboratory (JPL). Of more short-term interest is that we have been assisting JPL in their project by transferring knowledge that we have gained through our research and development efforts. In addition, our expertise in the areas of the TENEX computer system, the ARPA network, and computer language design has been applied to support the complex system programs required in the JPL project. Thus, by combining the pragmatic and theoretical aspects of robot development, we have created an approach which is grounded in realism, but which also has at its disposal the power that comes from looking at complex problems from an abstract analytical point of view.

## II. The "Robot" Computer Problem-Solving System

### A. The Robot and its World

The world which surrounds our simulated robot is diagrammed in Figure 1 on the following page. This figure, which is a computer-drawn plot, represents an aerial view of a section of "Martian" landscape that is nominally 400 feet square. It is a planar area which is partially bordered by the edges of three mountains and includes within it two hills and four craters (one a double crater). The plane (to which the robot is restricted) is randomly sprinkled with 50 rocks and 50 "unidentified objects" (which are small objects that resemble each other). Additionally, a single "instrument package" is drawn as a unique symbol which appears to the right of center in the figure.

This landscape is of course fictitious, but it does resemble lunar terrain and similar cratered portions of the Martian surface. The mountains, hills, and craters were designed by hand, but the smaller objects were distributed on the basis of computer-generated random coordinates. The "unidentified objects" are provided so that there can be another type of object which is more distinct from rocks than rocks are from each other. The "instrument package" is provided so that we may experiment with the

effect of having a unique object in the environment.

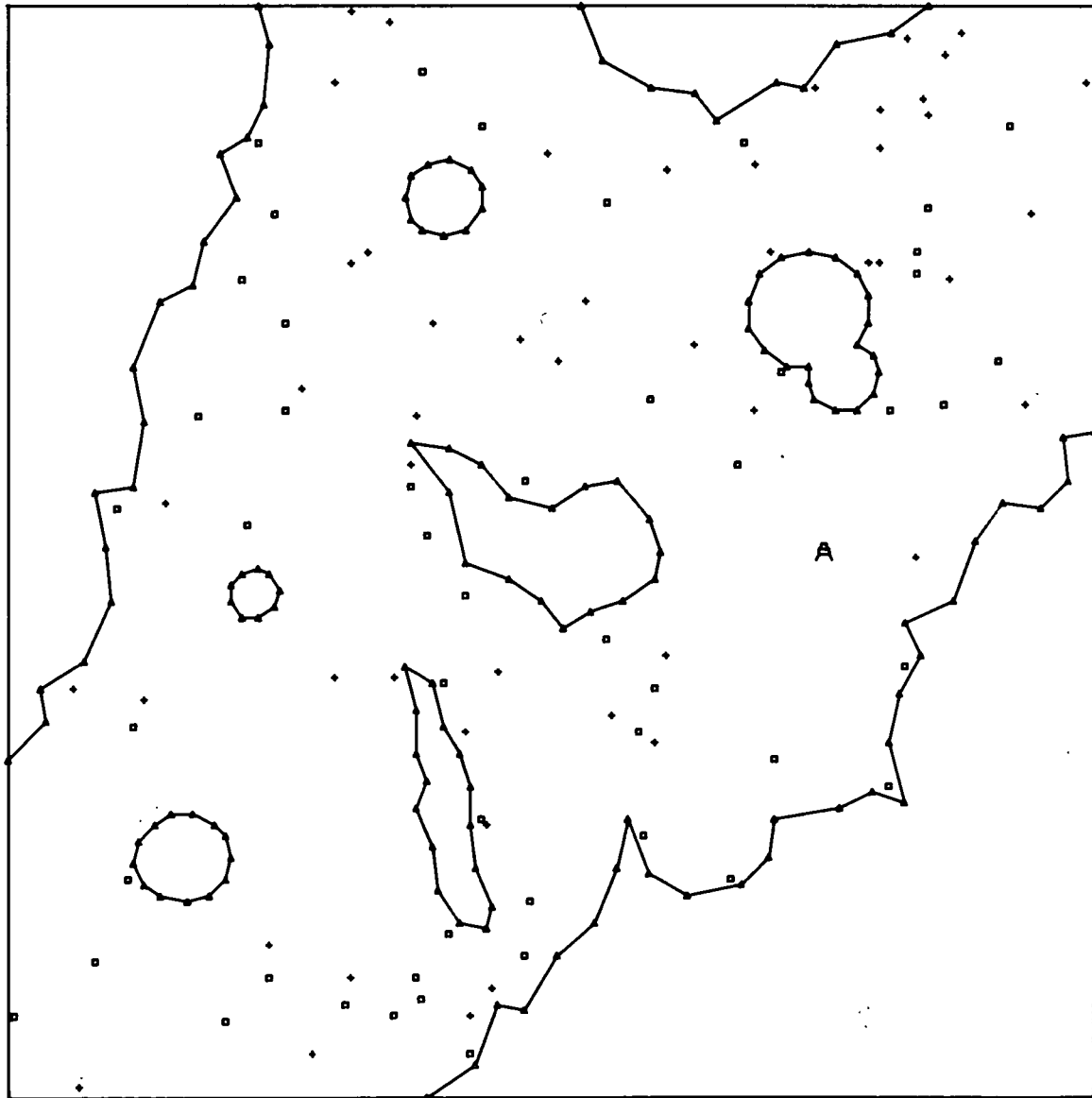


FIGURE 1: The Robot's World

The simulated Martian world provides an environment which is very rich visually, since it contains many objects both large and small, occurring in a variety of shapes and clusters. This informational richness, rather than any particular detail of its design, is the important feature of the simulated world as far as our study of robot perception and intelligence is concerned.

The robot itself is shown in Figure 2 on the following page. The robot is seen just above the instrument package, looking rather like a car with tailfins (the only reason for this shape is that it is easy to tell front from rear). In front of the robot is seen a projection of its visual field onto the landscape. We will discuss the robot's visual system at length in the next section.

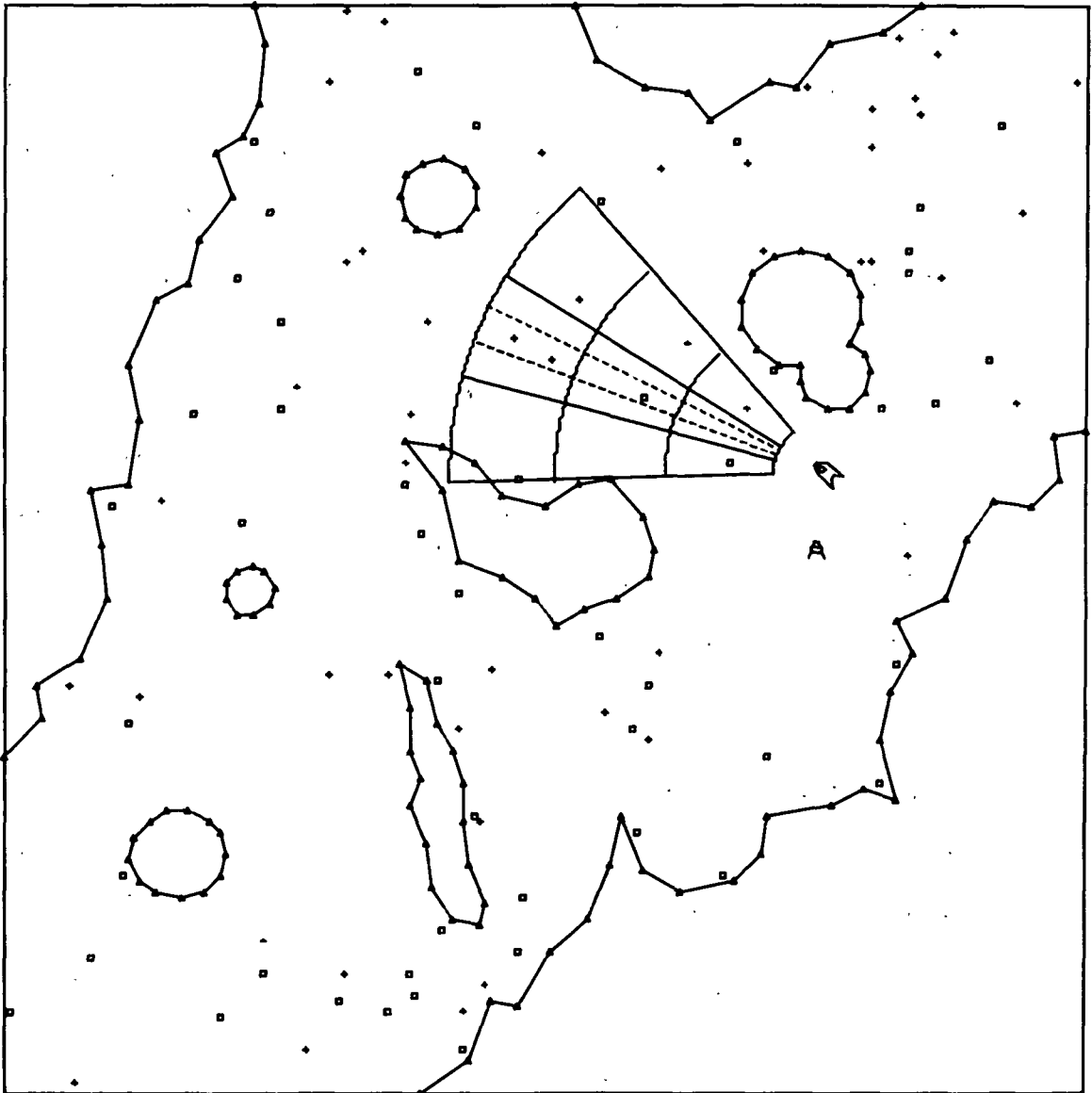


FIGURE 2: The Robot and its Visual Field

At the moment, we have chosen to restrict the robot to a fixed path, consisting of 328 short segments (averaging about seven feet long in scale units) which form an intricate tour of the given terrain. The reason for using a predetermined path at first is simply that the robot does not yet have enough intelligence to set out on its own course. In effect, we are holding it by the hand and guiding it along. Eventually, we do hope to permit the robot to roam freely in its environment, and we are designing our system to allow for that possibility. In the meantime, the fixed path that we have designed contains curves, loops, turn-arounds, double-backs, and so forth, so that it presents a great variety of perceptual challenges. The robot's body remains tangent to the path as it wends its way around the landscape.

The robot eventually will have some sort of speedometer, but for the past year all of our development effort has been concentrated on its single, very complicated sensory system: simulated vision. This simulation forms the subject of the next section.

## B. The Visual System

### B.1. Point Objects and Features

Our robot's only highly-developed sensory system is its monocular visual system. We should make it clear from the outset that we are interested in simulating the psychological or perceptual properties of a whole visual system, rather than in producing a model of the physical and sensory properties of the eye alone. (See Appendix A for a more elaborate statement of the approach we are taking here). That is to say, our main interest is in the robot's experiential learning and problem-solving abilities, and it is safe to suppose that these "higher" systems do not receive raw sensory data, but rather are fed information that has already undergone a certain amount of internal preprocessing. Therefore, our simulation of the visual system is not so much a model of the eye as it is of the early stages of visual perception.

To see what this means in specific terms, consider what happens when the eye looks at a continuous curve. We postulate that in both animals and machines, continuous input information from the world is broken down into discrete information by the lower levels of the perceptual system. That is, we hypothesize that even when a human looks at, say, a mountain, he does

not gather an infinitude of data, but rather his visual system automatically extracts a sampled set of data points that are either extremal (e.g. a maximum of curvature, the edge of a shadow) or typical of an area (e.g. a typical point in the middle of an arc of constant curvature or in the middle of an area of constant color).

Figure 3 on the following page gives a schematic example of how a continuous curve may be reduced to the set of its most typical and least typical points. This transformation represents the way that the robot sees the edge of a mountain, hill, or crater; that is, it sees a finite number of discrete points, called Terrain Feature Points. These are plotted as tiny triangles at the vertices of the mountains, hills, and craters in Figures 1 and 2. It is obvious that the density of this sort of sampling is much greater in human vision than in our robot model, but we feel that the principle is the same, so that the model is suitable to the purposes of our investigation.

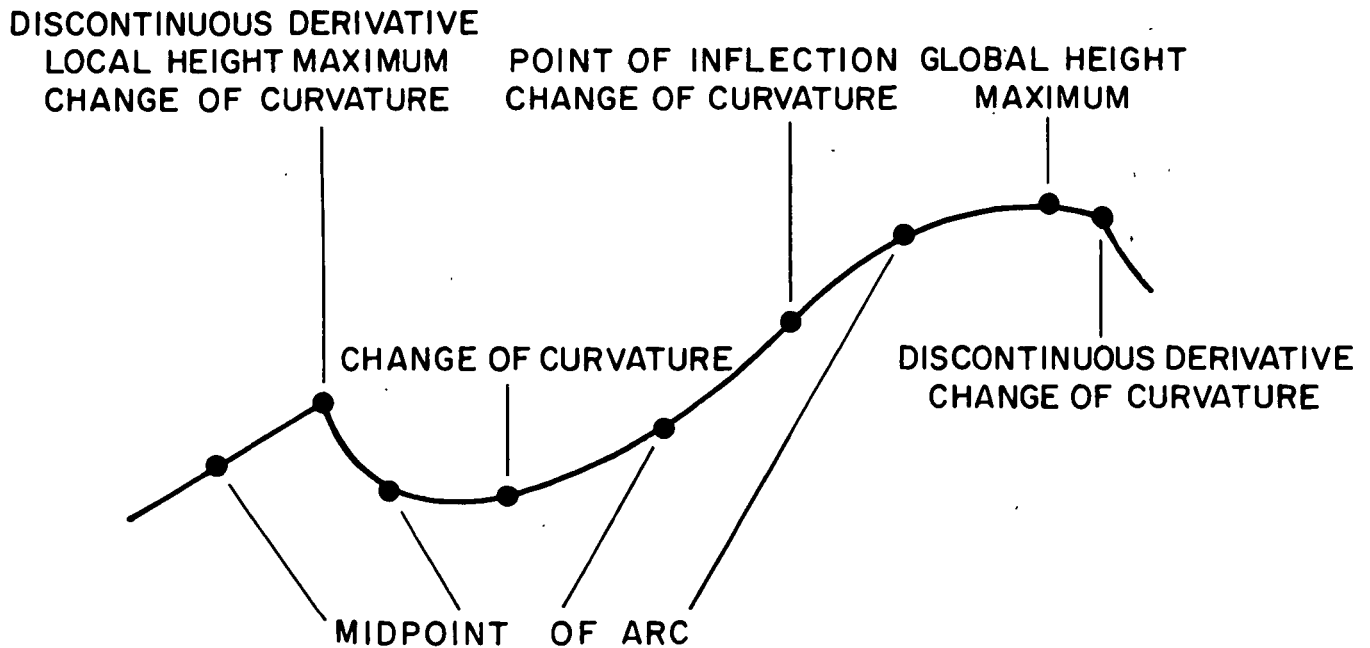


FIGURE 3: Feature Points Along A Continuous Curve

The mountains, hills, and craters in our robot's world are collectively called Terrain Objects. Terrain Objects have spatial extent, and are composed of many Terrain Feature Points, which themselves have locations but no extent. We were also interested in having objects which were themselves small with respect to feature extraction by the robot's visual system, so we introduced the rocks, "unidentified objects", and "instrument package" that are strewn about Figures 1 and 2. These Point Objects are all regarded as sizeless -- even the instrument package, despite the

visual field are independently variable). This means that the visual field, if projected onto the landscape, would have the "truncated pie-shape" which appears in Figure 2 and which is redrawn for clarity in Figure 4 below. The visual field is divided both radially and tangentially into thirds, and the central angular sector is divided again into thirds, giving a total of 15 subfields of the visual field.

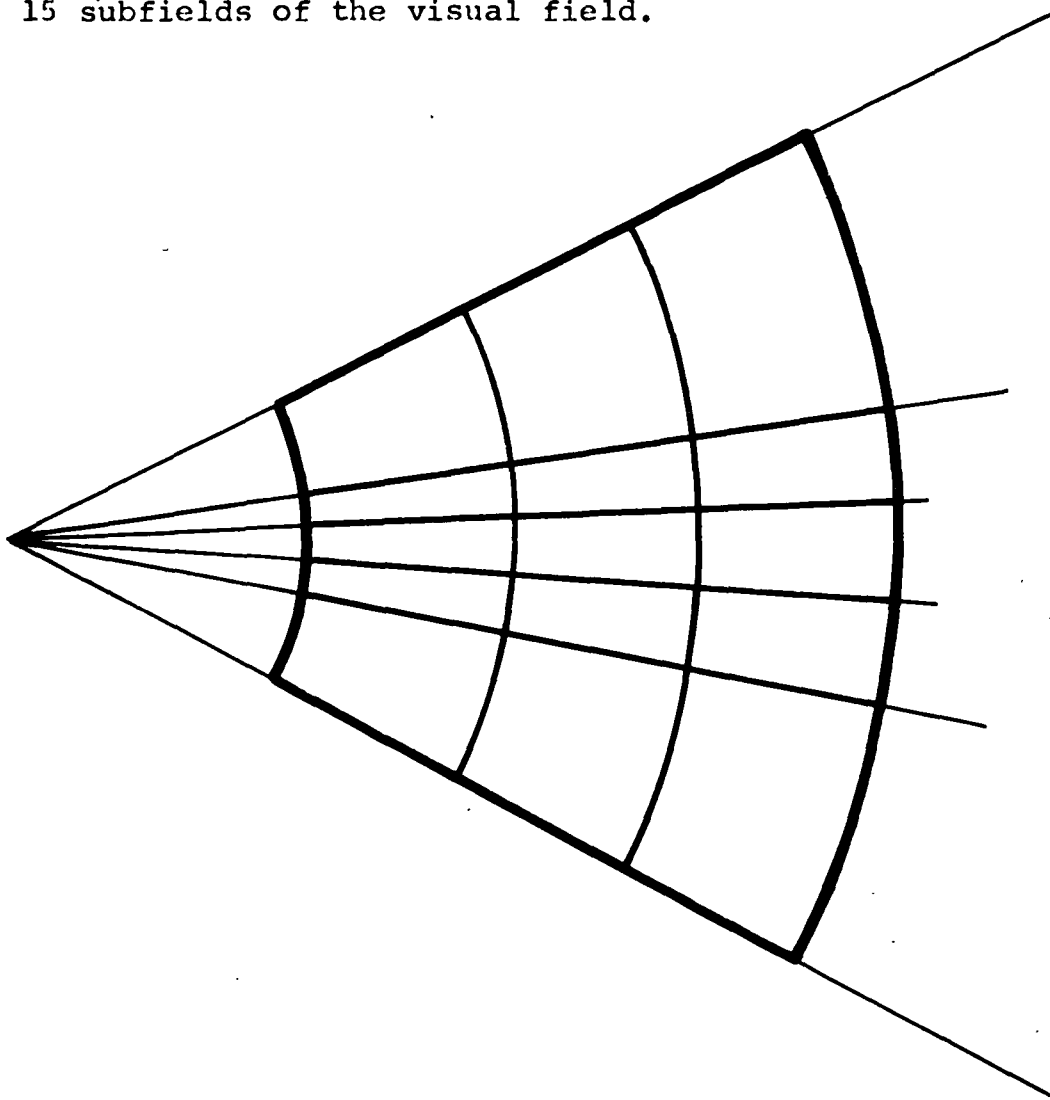


FIGURE 4: The Shape of the Visual Field

fancy way in which it is drawn by our graphical system. Clearly we are not trying to assert that rocks and so forth are infinitesimal; rather, we are trying to establish a distinction between objects which appear "large" to the visual system (the Terrain Objects), and those which appear "small" (the Point Objects). We feel that having both types of object in the world will give us more opportunities for investigating the process of visual perception than we would have using either class alone.

We may refer to the Terrain Feature Points and the Point Objects collectively as Feature Points. Then we may summarize by saying that the output of our simulated visual system reflects only the existence of Feature Points. The large Terrain Objects have been decomposed into Feature Points, and their existence must be rediscovered by internal cognitive processes. The bases for such processes are discussed in Section II.B.4 below.

## B.2. Vision

The robot's eye is focused on some certain point in space; the robot can see all Feature Points which are within a certain distance of the focus point both laterally and radially (the width and depth of the

Now, each Feature Point has associated with it a permanent list of visual feature-value pairs. The nature of the features depends on what type of Feature Point it is, be it Terrain Feature Point, rock, unidentified object, or instrument package. The values of the features are distributed over certain ranges, so that, for example, a rock might have the feature COLOR, with one of the possible values GRAY, BLACK, SHINY, etc.; the feature TEXTURE with one of the possible values SMOOTH, CRYSTALLINE, ROUGH, etc.; and so on for other features. The distributions of values are such that some features have a higher total information content (i.e., more nearly uniform distribution of values) than others; besides, some individual Feature Points are of course quite distinctive with regard to their total set of features, while some are quite commonplace. Furthermore, each feature has associated with it a "noticeability" number, representing the fact that some features are more striking than others (e.g., color is generally more readily noticed than is visual texture). (We are considering that the "noticeability" should actually be represented as a probability, since sometimes a glance at an object will reveal one of its more obscure features for no discernable reason.) Occasionally, some values of a given feature render it more noticeable -- e.g., the color red attracts our

attention (and the robot's), whereas the color green does not especially.

When a Feature Point falls within the visual field, the vision simulation calculates which feature-value pairs from that Feature Point are actually visible at the given moment, taking into account the following considerations:

- (1) The "noticeability" of that feature (or of that particular value, in special cases like the color red);
- (2) The value of a "bias" factor controlled by the problem-solving executive. This bias allows the visual system to "look for" a certain feature of the environment, such as the visual textures of things, and thereby be more sensitive to the designated features than it would be in its normal state;
- (3) The subfield in which the Feature Point falls. The closer to the center of the visual field the Feature Point is, the more of its features will be seen. The central subfield therefore corresponds to focused foveal vision;
- (4) The total size (area) of the visual field. The smaller the area of the visual field, the more features are seen from each Feature Point falling within the field.

This last property allows the robot to "focus down" its attention on a single Feature Point, thereby seeing it more and more clearly, while at the same time eliminating more and more peripheral distractions. Notice that this mechanism of "focal attention" is more a property of visual cognitive processing than of the

eye itself, but that our simulation treats the visual system as a whole, for reasons given at the beginning of this section. At any rate, this feature of focal attention in our model is extremely important in emphasizing the active nature of the perceptual process (the robot must actively focus on an object in order to see it well) and also in emphasizing the informational tradeoff that faces any limited-channel information-gathering system (it can look at many things casually or at a few things intensively, but not at many things intensively all at once).

The final result of vision is that each visual subfield reports the feature-value pairs that were seen within it. Note that these pairs are all lumped together in a list, regardless of how many individual Feature Points they actually came from: that is to say, the visual subfields represent the limit of the visual system's spatial acuity. Therefore, the notion of separable Point Objects is not given explicitly to the robot in its initial perceptual data, but rather the Point Objects must be reconstructed by later cognitive processes. These processes are discussed in Section II.B.4 below.

### B.3. Visual Occlusion

In our simulation model we have gone to a great deal of trouble to simulate the visual occlusion (blocking) of one object by another. That is, when a Terrain Object stands between the robot and some other object, the Terrain Object cuts off the robot's view of the more distant object. To be more precise, mountains occlude anything standing behind them; hills occlude anything except mountains; craters occlude anything except mountains, hills, and the instrument package; while Point Objects, being points, do not occlude anything. (This ordering might be thought of in terms of the height of the objects, but it must be borne in mind that as far as the robot's visual system is concerned, the world is seen as two-dimensional; that is, aside from their occlusion properties, Terrain Objects do not have any vertical features that are visible to the robot.)

The meaning of visual occlusion for the robot is demonstrated in Figures 5a and 5b, on the following two pages. In Figure 5a, we see a plot of all objects in the environment, with the robot's eye in a particular position. Figure 5b shows the robot's eye in the same position, but now we plot only those Feature Points that the robot can actually see. Notice that the

central hill is visible behind the double crater, but that the small crater is entirely occluded by the central hill. The mountain ridge to the left of center is visible behind both crater and hill. The long hill below center is half hidden behind the central hill. Notice too that objects can occlude parts of themselves; in particular, the rear surfaces of the double crater and of the central hill are occluded by their own front surfaces. This also applies to concavities in a Terrain Object, such as the single point in the mountain ridge to the left of center that is hidden by a concavity, and the four mountain points in the lower center that are hidden behind the protruding escarpment.

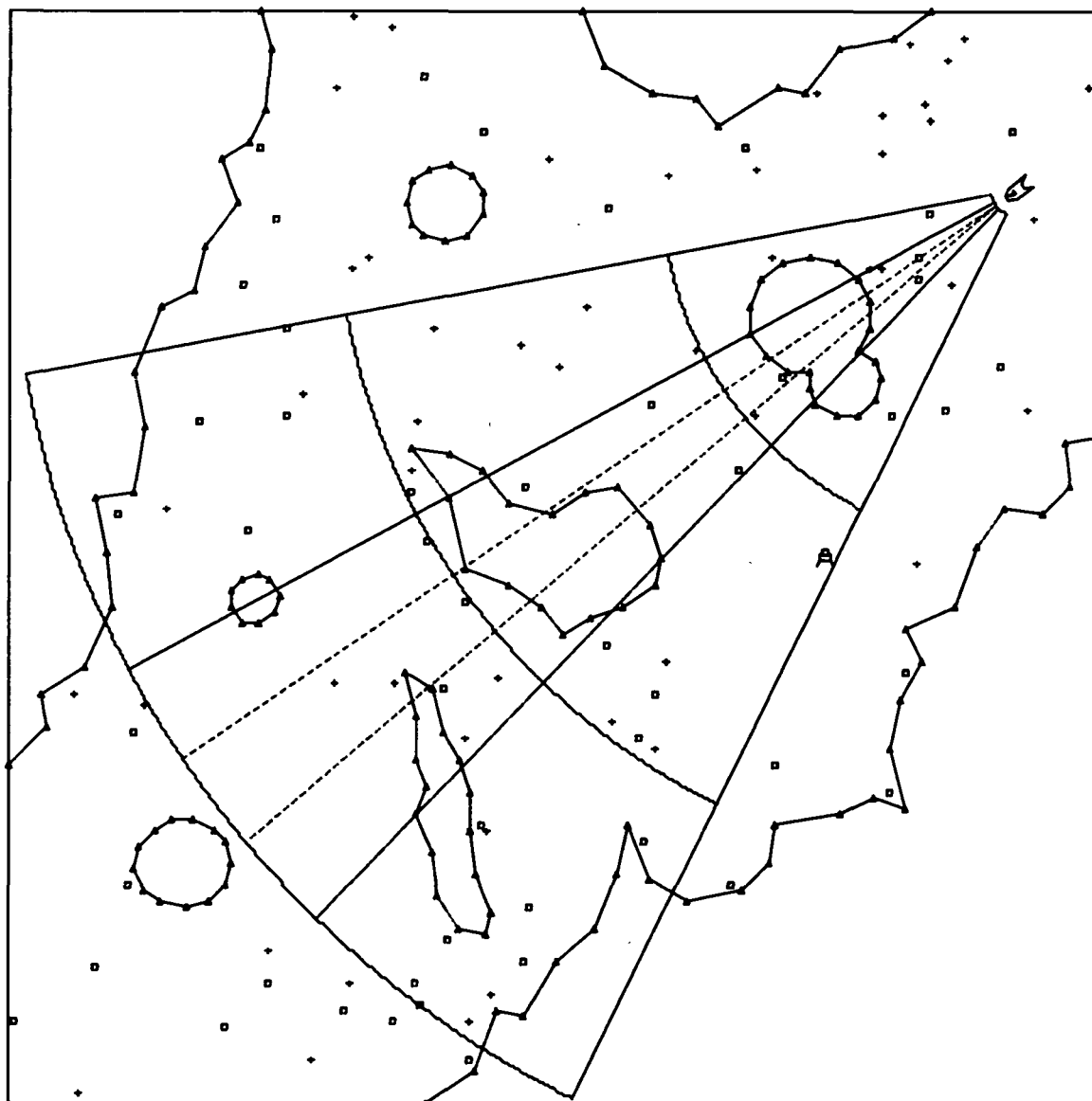


FIGURE 5a: Showing All Feature Points

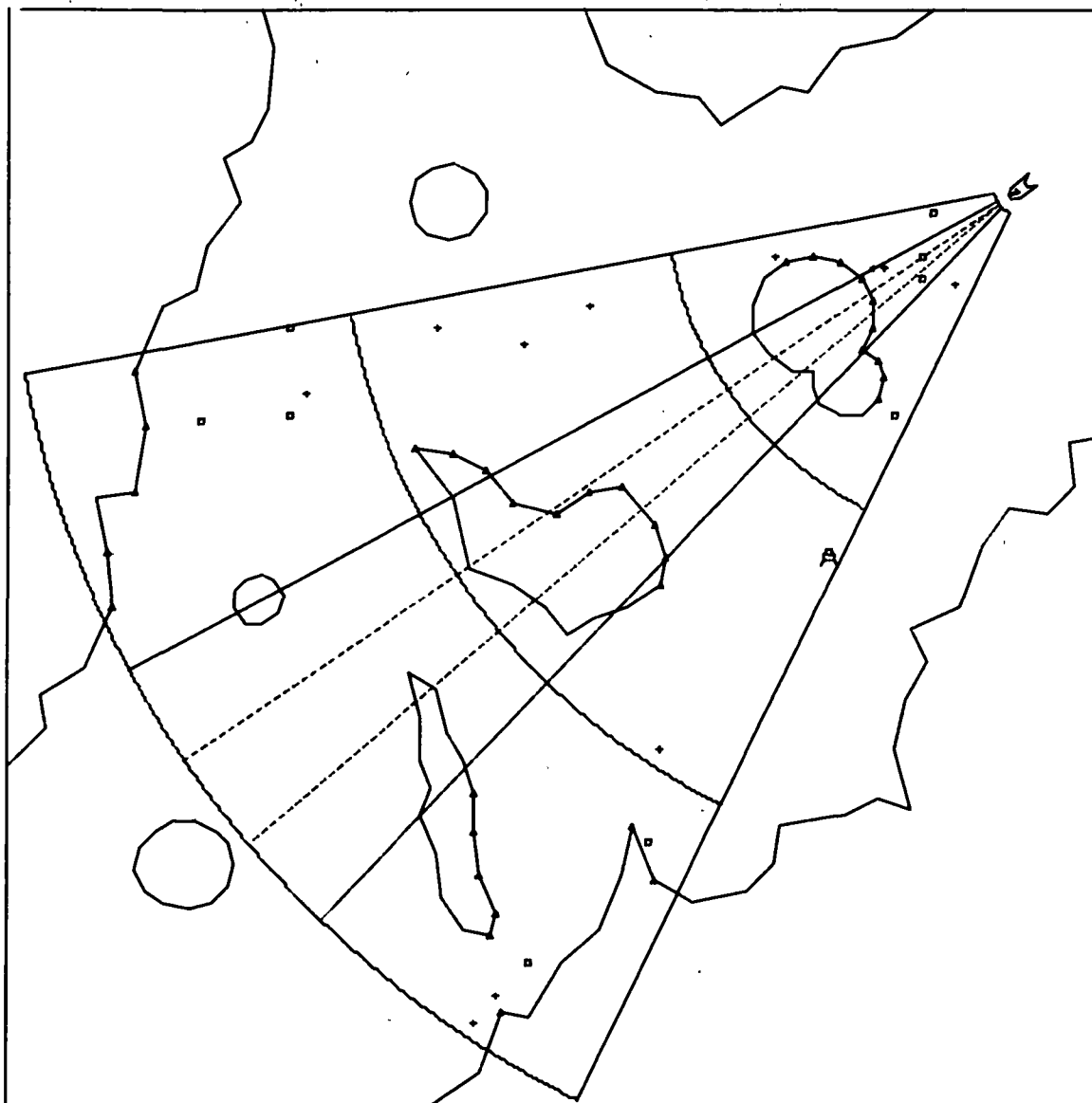


FIGURE 5b: Showing Only Those Feature Points  
Actually Visible

The algorithm that computes which Feature Points are visible and which are occluded, since it is somewhat complex, is attached as Appendix B to this report. Our motivation in including visual occlusion in our simulation was partly to increase the physical verisimilitude of the model, but more importantly to increase its psychological verisimilitude. Visual occlusion has important interactions with the perceptual processes we wish to study, as we will indicate in the following section.

#### B.4. Object Reconstruction

Recall from Section II.B.2 that the output of the robot's visual system is merely 15 lists of visual feature-value pairs, one list for each subfield of the visual field. It is important to note that this information is not encoded in terms of objects; even Point Objects do not appear explicitly in the output of the visual system although they directly give rise to some of the feature-value pairs. Therefore, in order for the robot to arrive at the concept of "object", it must apply higher cognitive processes to the perceptual input that it receives. Although we have not programmed these processes as yet, we will discuss their nature below because they are important for an

understanding of the level at which the robot's cognitive operations begin; also they are relevant to some other aspects of our simulation presented in Section II, especially visual occlusion and tracking.

### Point Objects

The beginning of the notion of object is the list of feature-value pairs that are reported by the visual system as coming from a certain subfield of the visual field. This set, which we might call a "clump" of features, at least carries the assurance of a certain degree of spatial proximity among its elements. When the robot watches the same locality over an extended period of time (as it does in the case of tracking discussed in the following section), a clump may change in membership and increase or diminish in size, owing to the fact that the robot or its eye may have moved so that the visual subfield boundaries do not fall across the landscape in precisely the same way that they did formerly. By noticing common "factors" among various clumps of features, the robot may discover irreducible clumps -- namely those which come from single Feature Points, among which are Point Objects.

Another means of isolating the individual Feature Point is to focus down the eye on it until all other Feature Points are excluded. We wrote a program that did this in last year's version of the robot model (in the city-street environment), and this program should transfer easily to the new environment.

Thus, it is only by active effort and a considerable amount of bookkeeping that the robot can come to obtain even such simple facts about its world as the identity and properties of the Point Objects surrounding it.

#### Compound Objects

A Compound Object is a set of Feature Points. Presumably, some sets of Feature Points will become of interest to the robot for varying reasons: the points may be closely related spatially, or they may share some common feature (e.g. three orange rocks close together), or they may be distinctive as a group (e.g. there are 50 rocks and 50 unidentified objects in the world, but only one case where a rock and an unidentified object lie side by side), or they may have some functional meaning to the robot (e.g. as navigational landmarks).

These criteria are manifold and quite subtle, but the discovery of useful Compound Objects also requires a vast amount of bookkeeping and statistical work which is more messy than subtle. In addition, the question of forming concepts for Compound Objects is inseparable from the question of recognizing them upon returning to them after a period of absence. This is quite equivalent to the general "pattern recognition" problem, and we have discussed its particular difficulties at length in Section V of last year's summary report, Bolt Beranek and Newman Report No. 2316.

In a word, we regard the discovery and recognition of Compound Objects as processes which are among the primary research targets of our whole investigation.

### Terrain Objects

Terrain Objects are all Compound Objects, but they have one important additional property: they occlude objects standing behind them, and indeed they occlude their own rear surfaces. This means that besides the above-mentioned criteria for discovering Compound Objects, the robot can use occlusion as a means of detecting the existence of Terrain Objects. Indeed, as the robot drives around a Terrain Object (if it has the

ability and initiative to do so), the Terrain Feature Points appear and disappear in a definite order (although not necessarily in a strict linear order if the object has concavities, as most of the Terrain Objects do). In general, one would think that the concept for a Terrain Object should end up being stronger than that for a Compound Object which consists, say, of just a clump of rocks. Notice, however, that the recognition problem -- which is difficult in any case -- is further complicated by the fact that the robot can never see more than half of a hill or crater from any one vantage point.

Terrain Objects will also influence the perception of other Compound Objects by occluding parts of them from the robot's view. They also greatly affect the process of tracking, in which the robot attempts to keep an object in sight while moving. Indeed, the whole process of navigating around obstacles toward goals is tied in with the visual and physical impenetrability of Terrain Objects. We are not sure yet how all of these factors fit together, but these are some of the questions that we are most looking forward to exploring.

To summarize the point of this section, our robot's visual system, although it does a certain amount of "preprocessing", actually presents the robot's cognitive system with a very primitive level of information. In particular, this information does not explicitly report the existence of objects, be they Point Objects, Compound Objects, or Terrain Objects. Therefore, in order to discover and recognize the various objects which populate its environment, the robot must perform quite a variety of information-processing procedures on the data that it receives from its visual system. Some of these procedures are in fact quite elusive and constitute major research questions in the development of robot intelligence.

### C. Tracking

The most advanced behavior of our robot at the moment is that it can track an object -- that is, as the robot moves it can keep its eye fixed on a chosen object on the basis of visual information. This behavior is not new to our robot; in fact it was transferred directly (with only a minimum amount of reprogramming) from last year's city-navigation model. Last year's program is described at some length in Section IV.B of our previous summary report, Bolt Beranek and Newman Report No. 2316. Our purpose here is mainly to present a diagrammatic example of the tracking process and to make a few additional comments.

Figures 6a through 6k on pages 29-39 show the robot tracking the instrument package while moving on a curvilinear path. Initially, in Figure 6a, the instrument package is in the right-outermost subfield of the robot's visual field. By Figure 6b, the robot has refocused its eye so as to bring the instrument package near to the center of the field (the increment between two successive frames within Figure 6 is a single quantum in the discrete time of the robot's world). As the robot moves and turns, it constantly readjusts the parameters of its visual field so as to retain the instrument package near the center.

Note that the instrument package is often not precisely centered, as especially in Figure 6i. This is because the tracking program operates by saccades (jumps) which are only approximate predictions of where the eye should be turned, given the expected future position of the robot. There is an important general principle here: it does not matter if the robot's actions are imperfect, so long as it has feedback mechanisms which can correct its errors. This principle is vital to the survival of robots, animals, and humans, since none of us can count on doing things perfectly the first time. In this case, the robot's continued visual contact with the instrument package is in itself the feedback required to correct inaccuracies in the positioning of the eye.

The tracking episode ends in Figure 6k, in which the instrument package has gone out of sight behind the lower extremity of the central hill. We were actually surprised when the tracking process reached this state and the instrument package disappeared from the display screen. This fact suggests three comments to us: first, that our occlusion algorithm is a lot more alert and sensitive than we are; second, that if we had not had the display to examine, we might well have spent hours looking for a nonexistent "bug" to explain the disappearance of the object being tracked; and third,

that the robot perhaps should have been a little surprised too.

In fact, at its present primitive stage, the robot has no means at all of comprehending the disappearance of a tracked object. Indeed, as we tried to indicate in the previous section, there is a fairly subtle relationship between the process of tracking and the very process of identifying an object. This relationship is especially complicated by the possibility of occlusion of one object by another. These issues will be further explored as we make the tracking procedure more sophisticated, and the robot more intelligent overall.

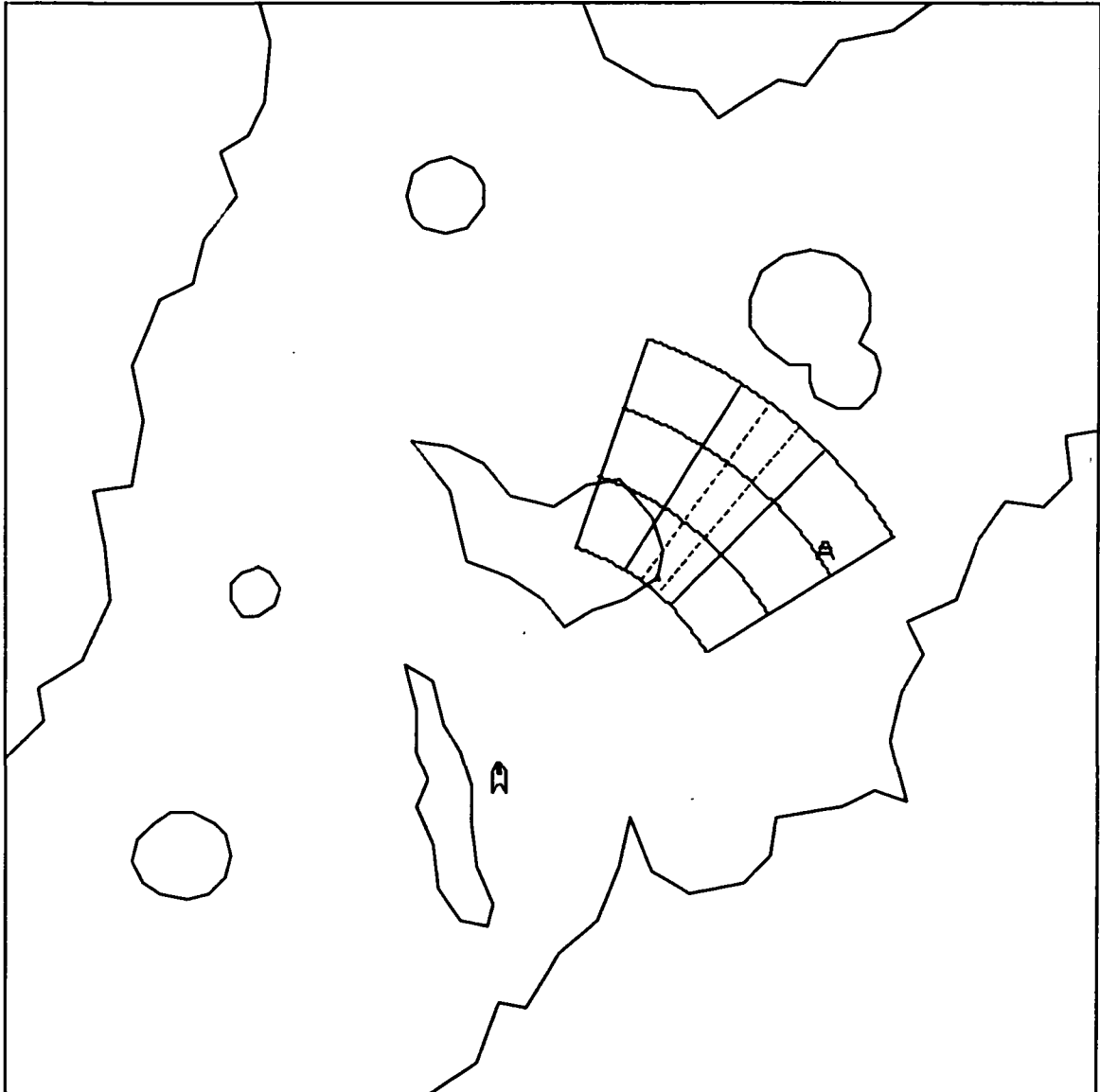


FIGURE 6a: Tracking

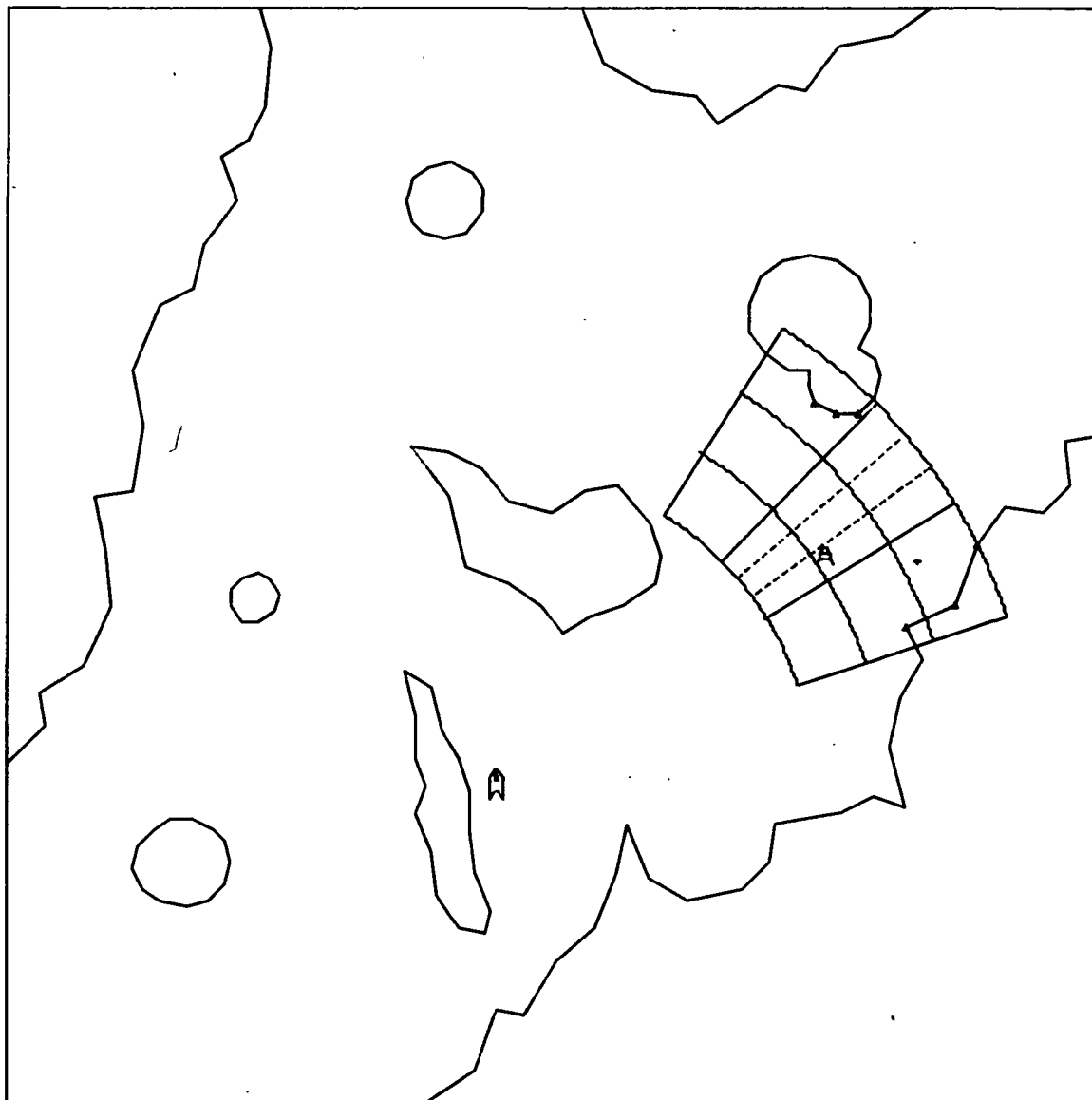


FIGURE 6b: Tracking

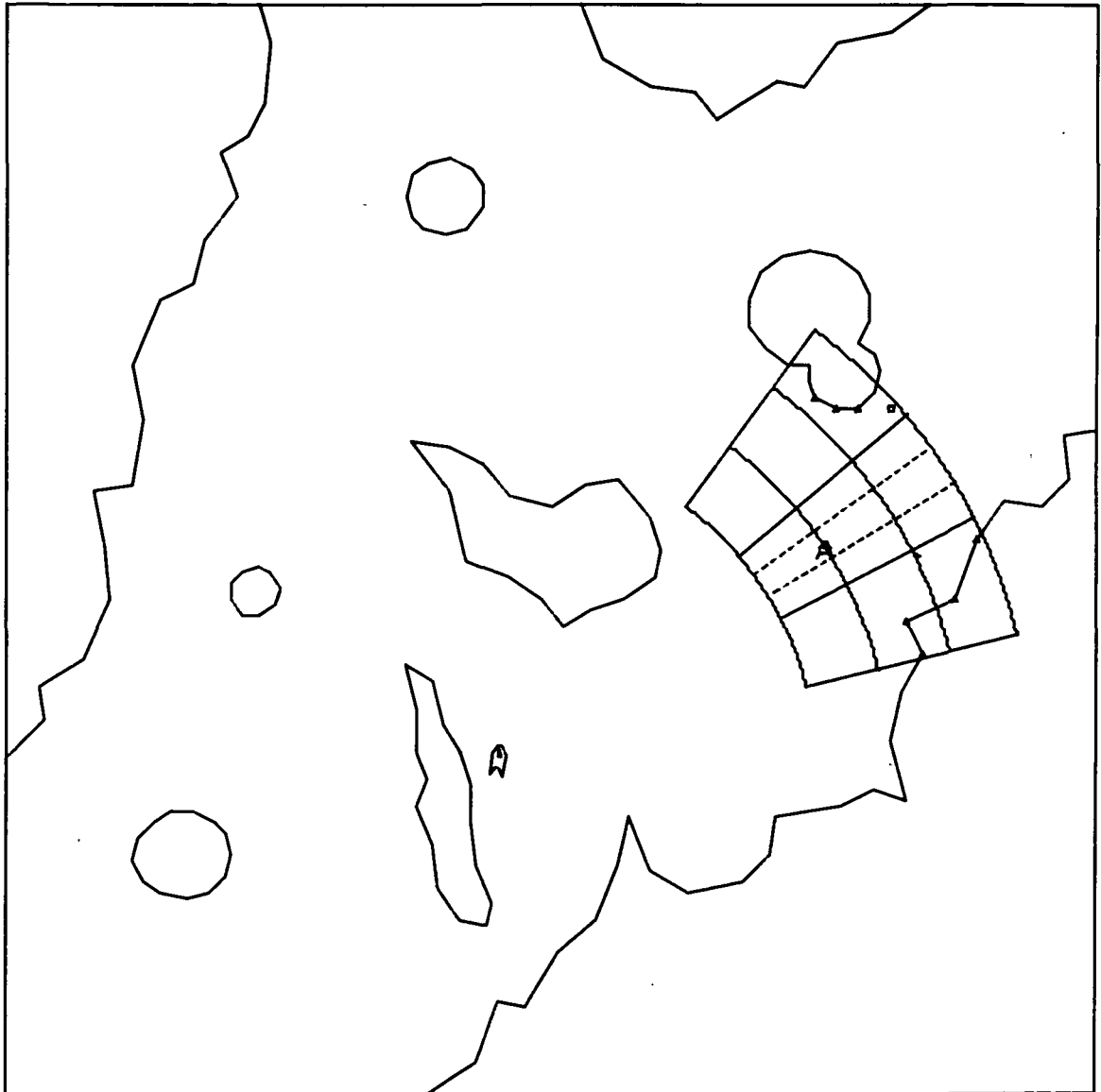


FIGURE 6c: Tracking

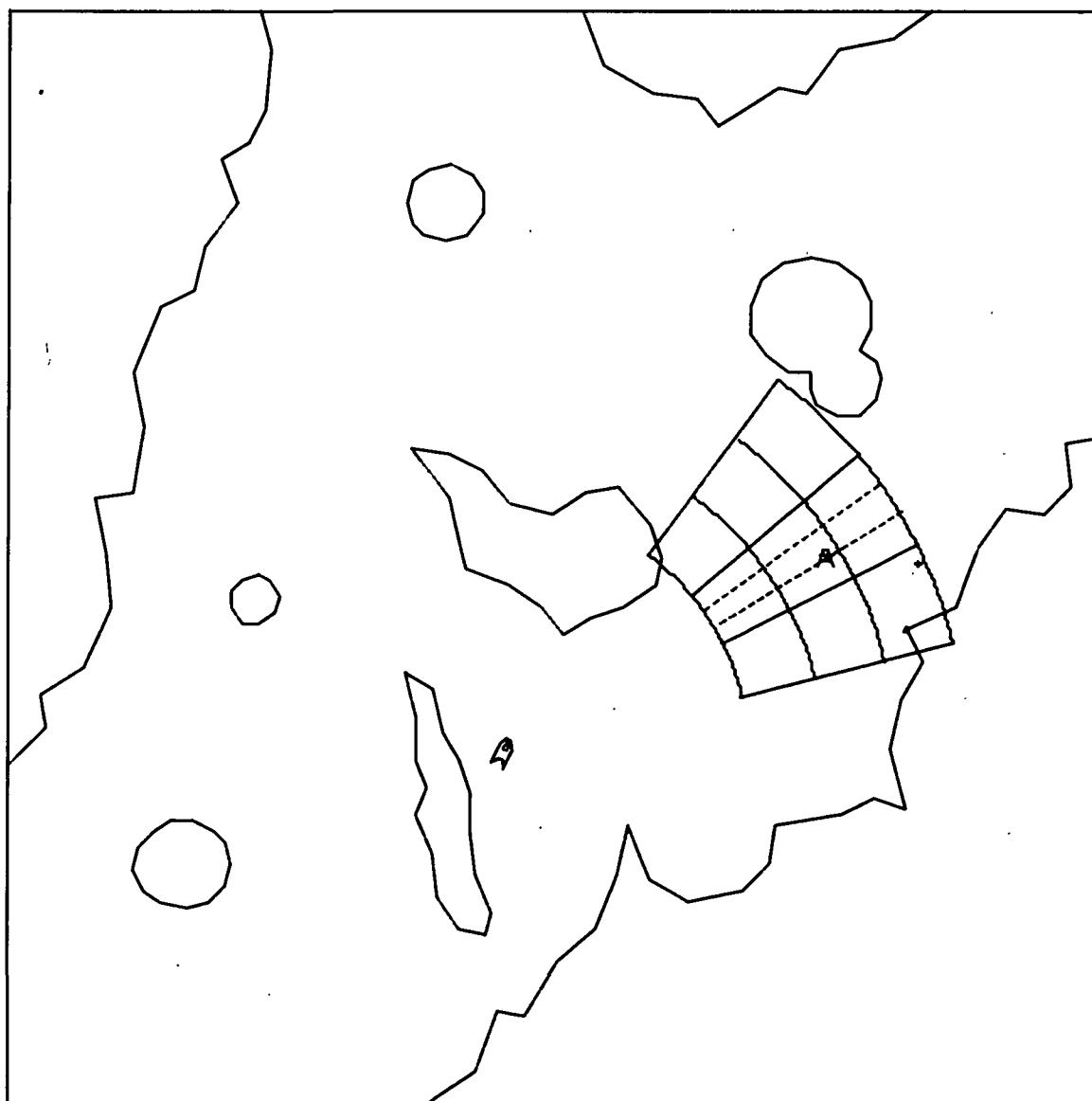


FIGURE 6d: Tracking

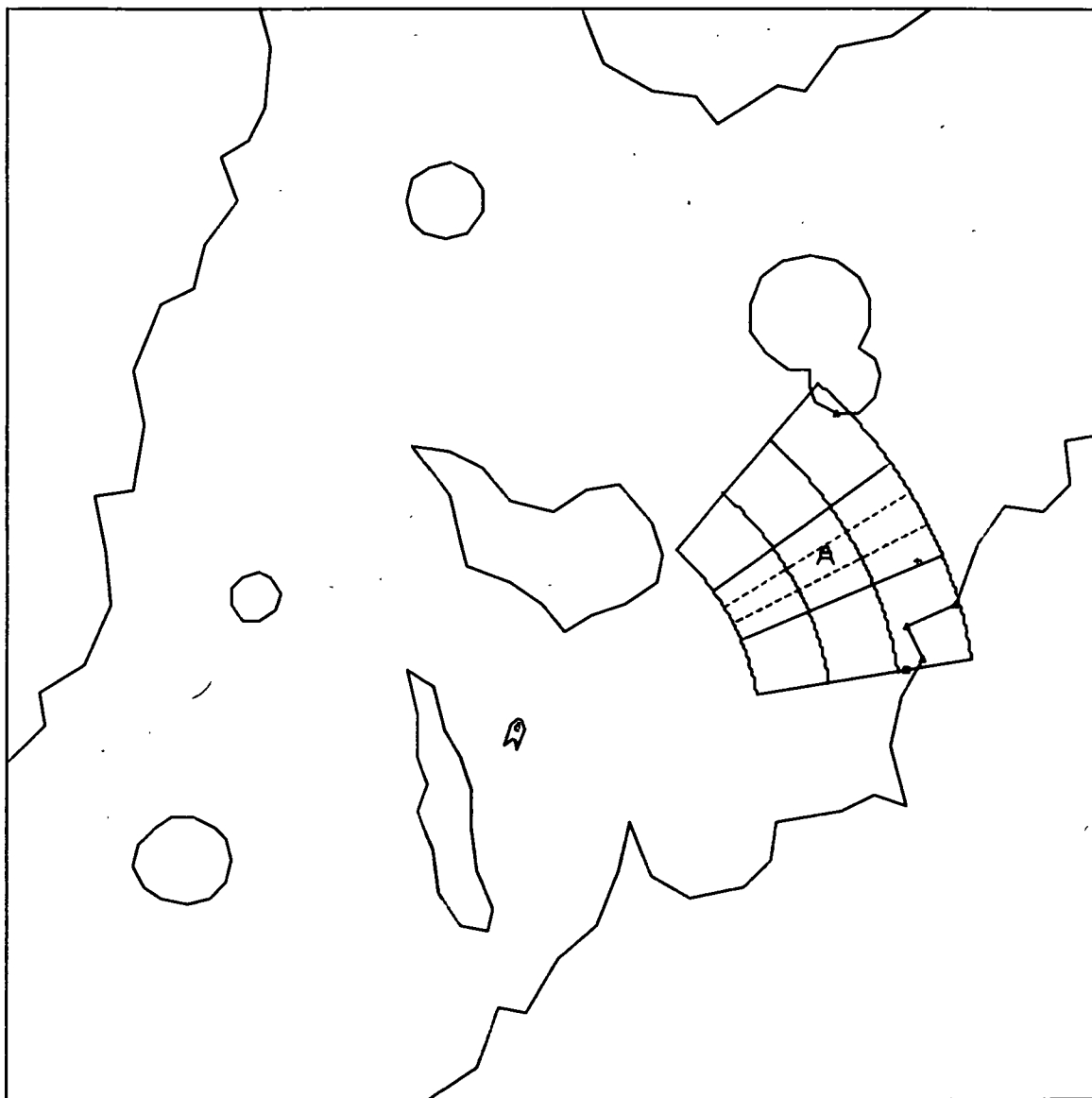


FIGURE 6e: Tracking

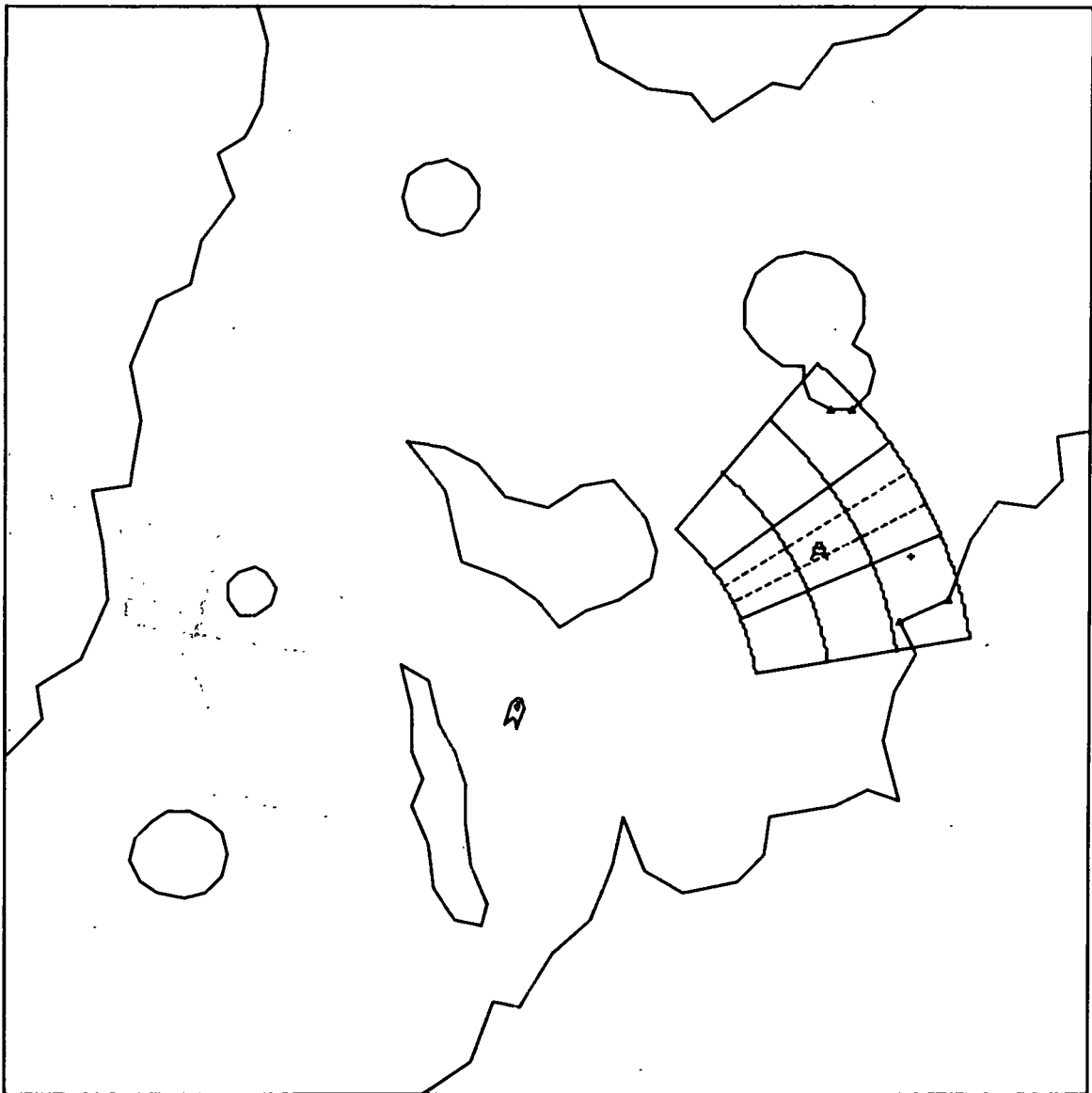


FIGURE 6f: Tracking

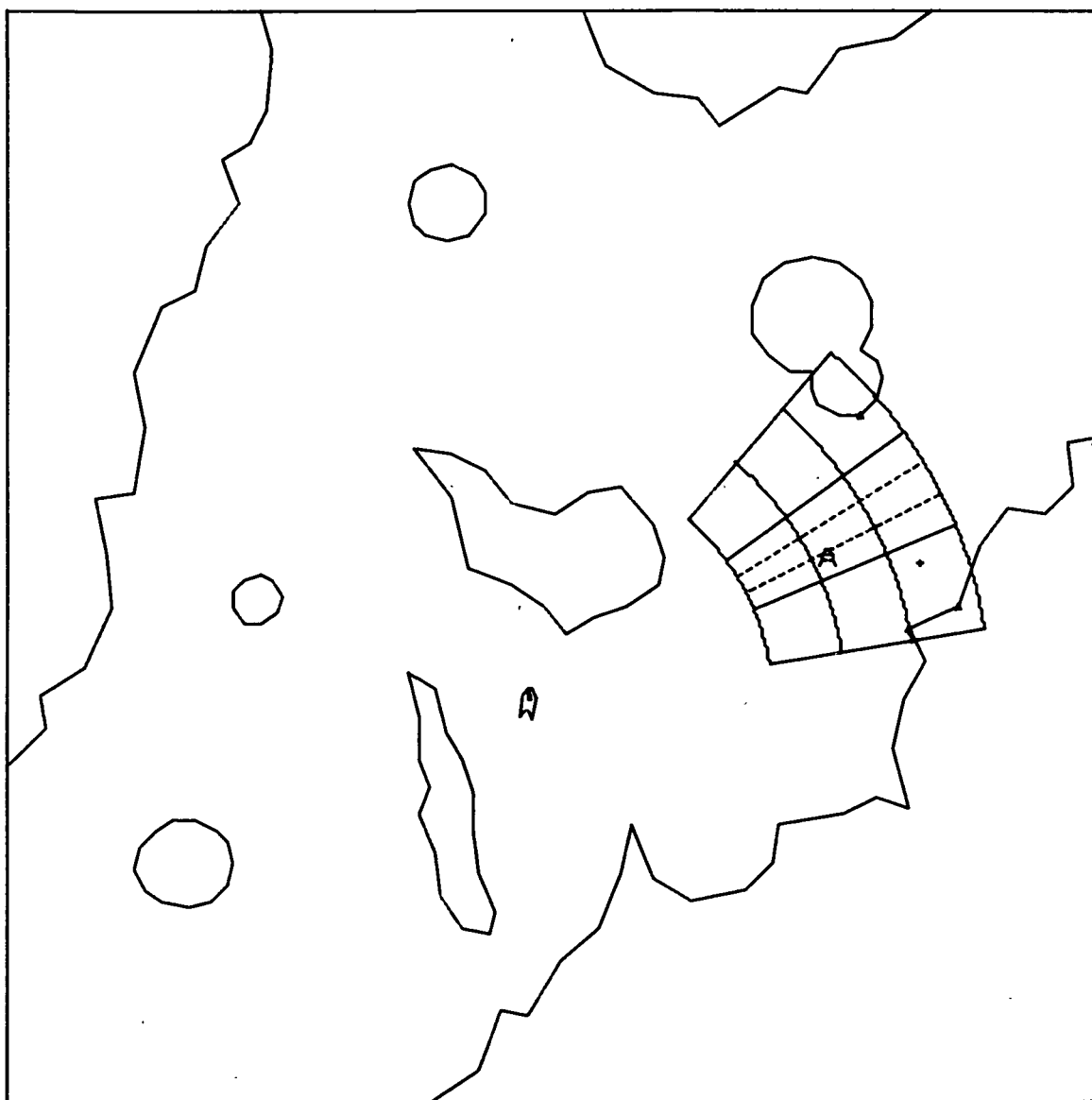


FIGURE 6g: Tracking

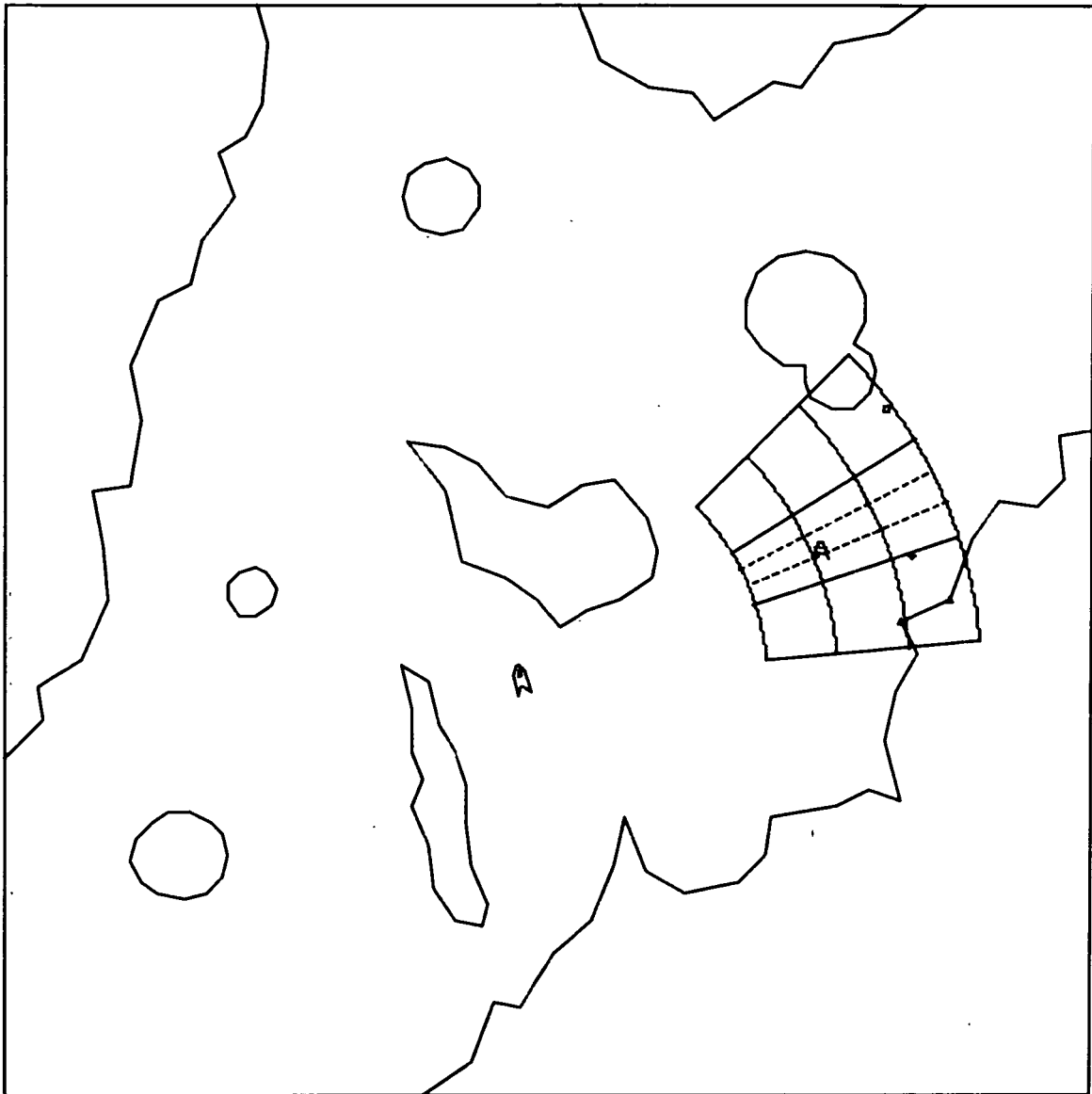


FIGURE 6h: Tracking

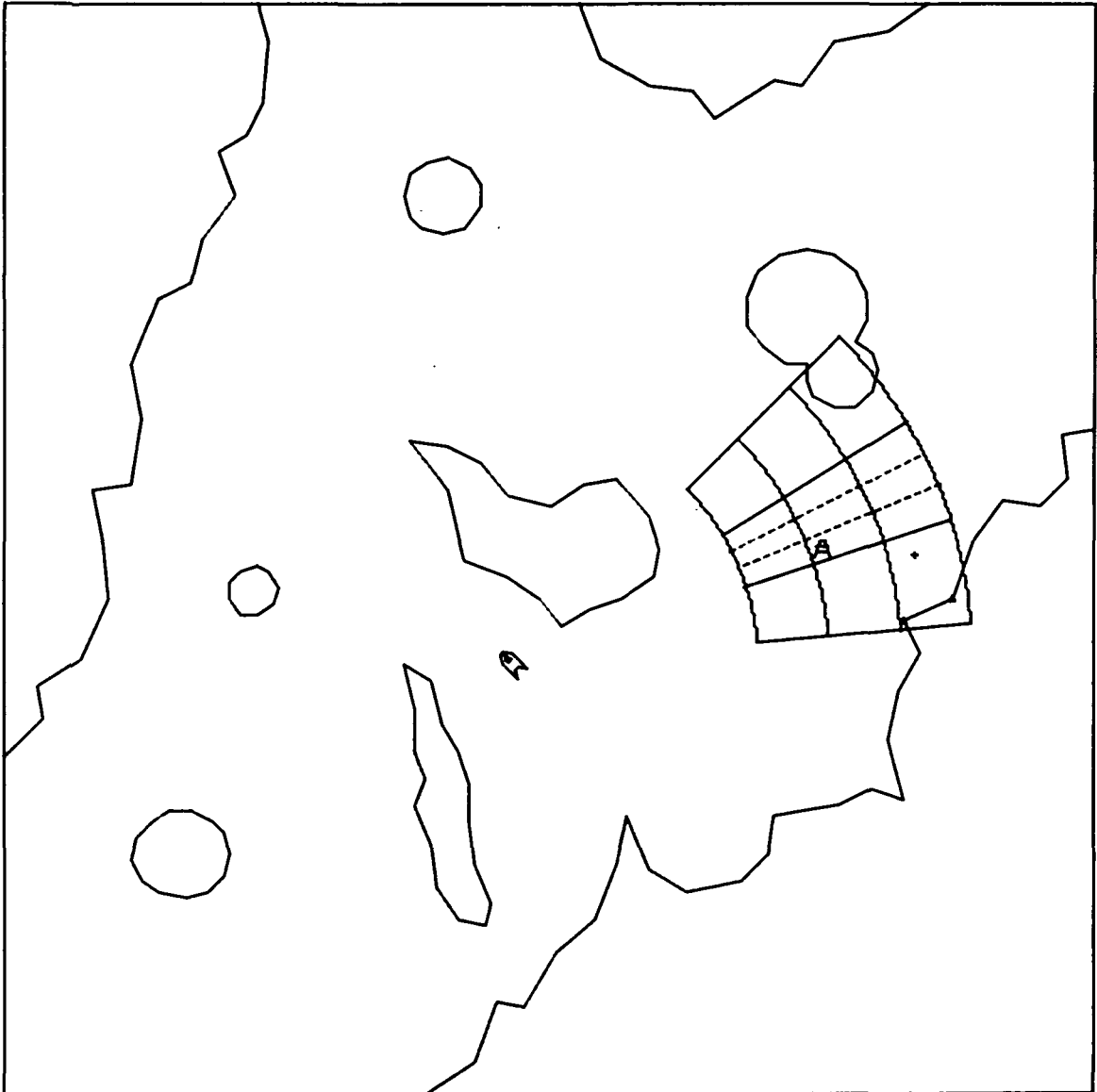


FIGURE 6i: Tracking

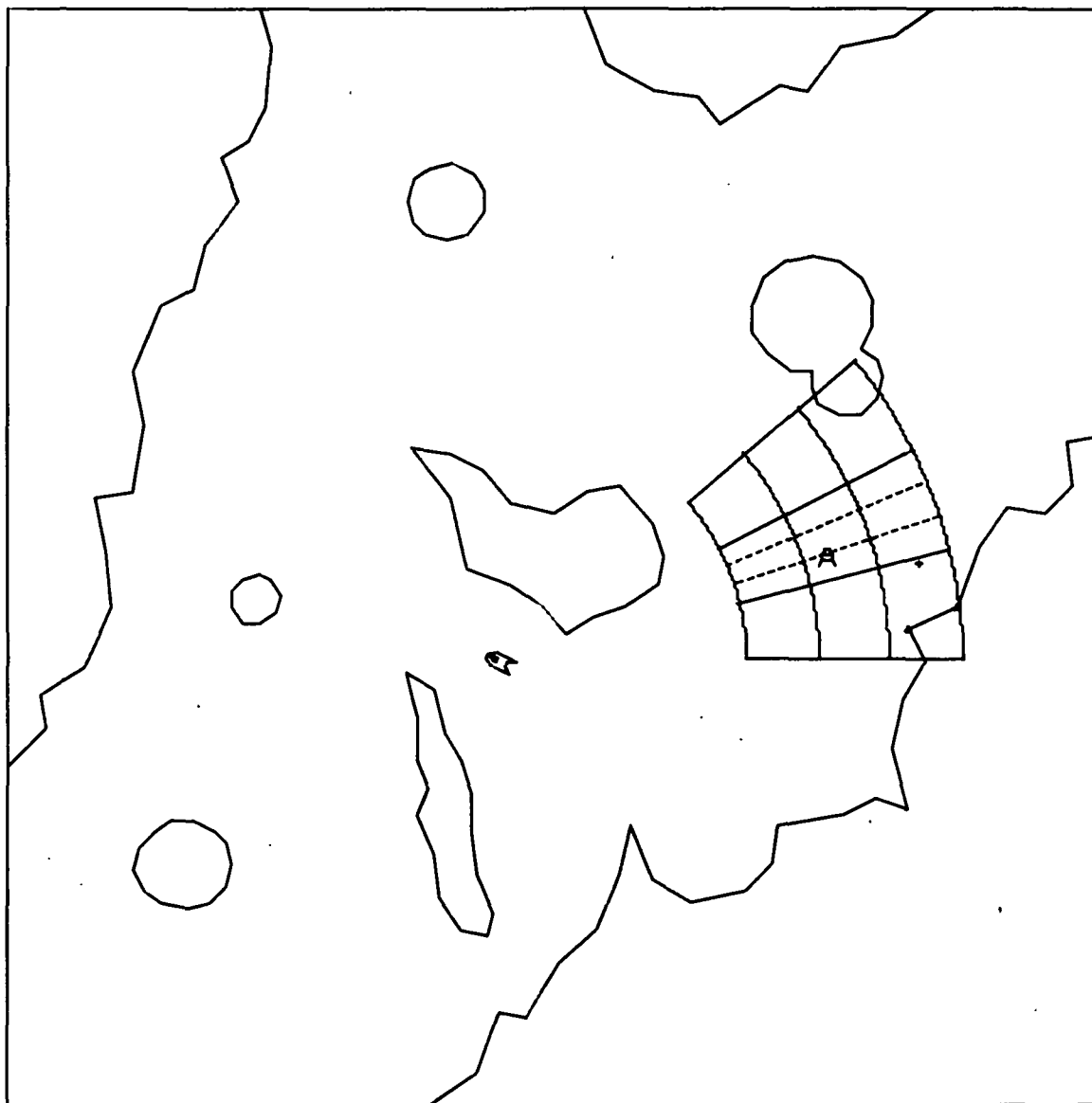


FIGURE 6j: Tracking

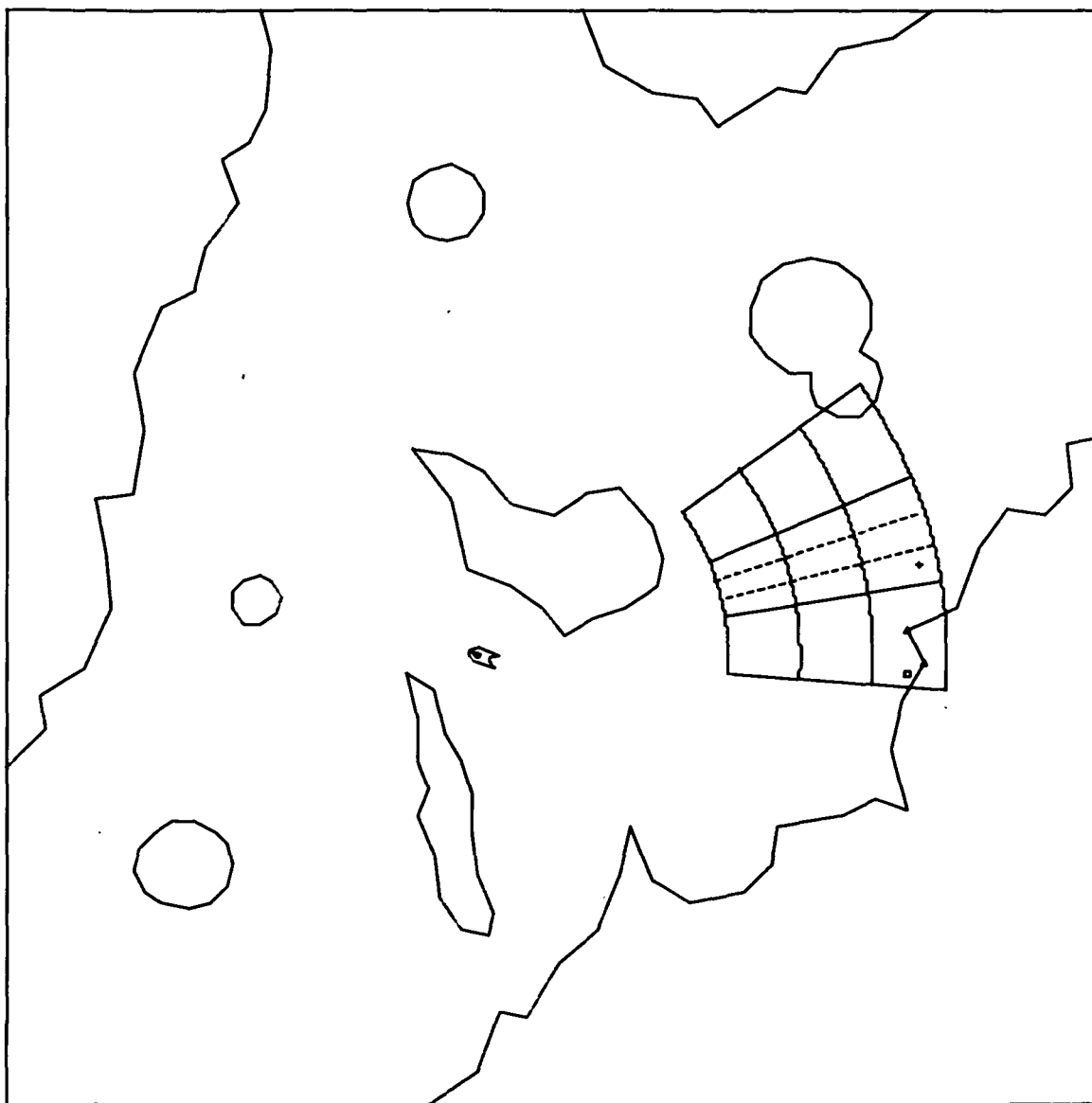


FIGURE 6k: Tracking

#### D. Robot Display

As the robot problem solving system grew in both size and complexity, it became increasingly difficult for a human to grasp all the ramifications of the robot's actions. For this reason, we developed a graphic display system which dynamically displays the simulated world, the robot, and information indicating the current state of the robot's problem-solving activity.

This visual simulation is run on BBN's Imlac Graphic Display system consisting of a mini-computer and a display processor connected to a cathode ray tube. The BBN-developed Time Sharing Imlac Monitor, TSIM, is used to share the Imlac processor between a simulated teletype and our robot-world display program. It also allows communication with the existing robot problem-solving system which runs as a cooperating process on the BBN TENEX computer. Additionally, it interacts with a TENEX program that allows the plotting of anything that appears on the display screen.

To ensure that the display is a useful tool in the design and debugging of the robot system, we have implemented the following features:

- 1) Since there is a large amount of information which is potentially desired, but which is not always needed, options are provided to minimize the clutter on the display screen. For example, the Feature Points in the world can be displayed or turned off by throwing a switch on the IMLAC graphic computer console.
- 2) To enable us to gain greater insight into the problems facing the robot, an option is provided so that only those Feature Points actually seen by the robot are displayed. The results of this option can be seen by comparing the two diagrams illustrating occlusion in Section II.B.3 (Figures 5a and 5b).
- 3) Provision is made so that any part of the display can be enlarged.
- 4) Processing is divided between the IMLAC and TENEX computers so as to attain a balance between maximizing real-time response and minimizing display flicker and blinking. Also, the bandwidth required for communication between the IMLAC and TENEX is held to a minimum for the same reasons.
- 5) A "mouse and bug" -type indicator is used to point to any part of the display. This eliminates the confusion caused by parallax when two or more people are watching the display.

These features have resulted in a display that is not cluttered except for those brief moments when we wish to display all available information. Response time to local (IMLAC) interactions is real-time and the response time from TENEX interactions is not materially affected by the amount of information transmitted

between the machines.

Most important is that the display has proved to be a tremendous asset in the development of the robot system. An interesting example of this occurred while we were hooking up part of the vision algorithm, when the robot turned its eye to an unexpected place. Because the position of the eye relative to the robot was visible on the display and because we knew where the eye should have been, it was immediately obvious that we had a problem. If we had had only printout information, as previously, it might have taken several hours to diagnose the same problem once we even realized that something was wrong. With the display, diagnosis was completed in about two minutes!

### III. Assistance with JPL Robot Project

Our assistance to the NASA-supported JPL robot research program has primarily been a consulting role concentrating on four tasks:

- a) The conversion of the SAIL programming language to run under the TENEX monitor, SAIL, originally written by Stanford University to run on a DEC 10/50 computer system, is being used by JPL for programming the higher level functions of the robot project. Since JPL plans to use a TENEX/PDP-10 computer system, conversion is necessary.
- b) Providing assistance to JPL in configuring their IMLAC computer system and furnishing software to support it as a sophisticated display terminal to be used in conjunction with the JPL robot project.
- c) Providing consultation on the ARPA Network, JPL is currently dependent for computer services on remote PDP-10 computer systems which are accessible via the ARPA Network.
- d) Consultation on artificial intelligence research.

Early in our assistance role we met with JPL and NASA personnel both in Pasadena and in Cambridge and then began actively assisting with the JPL robot project.

## A. Initial Planning Efforts

During the week of September 10, 1972, we visited Pasadena and attended the JPL Robot Research Program Review, attended the first NASA Remotely Manned Systems Conference, and met with JPL and NASA personnel to discuss augmenting the BBN-NASA contract to include providing assistance with JPL. By November 16, 1972, we distributed to JPL a JPL-BBN work agreement statement. This statement was the result of the initial meeting and several subsequent meetings and phone conversations. The text of the agreement follows:

- (I) Convert the SAIL programming language so that it functions under the TENEX operating system for the DEC PDP-10.
- (II) Provide consultation on ARPANET Technology.
- (III) Provide consultation on acquisition of hardware features and hardware/software trade-offs for JPL's IMLAC display system. If JPL's IMLAC is purchased with the required features, furnish JPL with a copy of the BBN IMLAC Monitor code.
- (IV) Assist JPL in the field of A.I. research, by providing consultation on world modeling, storage and manipulation of knowledge, and man-robot communication.

## B. Conversion of SAIL to TENEX

The SAIL Programming language has been used to define the complex control system for the mechanical arm which the JPL robot project intends to use. This language had been implemented to run on the DEC PDP-10 computer under the 10/50 monitor system. Since JPL is planning to use the TENEX monitor system, it was necessary to convert SAIL to run under this monitor.

In September of 1972 we made a fact-finding trip to Stanford University to determine what would be involved in converting SAIL for TENEX. We attempted to estimate the scope of the conversion task and to consider methods for incorporating new versions of SAIL into TENEX as they are created, and we brought back extensive documentation and several computer tapes with source files from Stanford. We then undertook the conversion task which is described in the following sections.

### B.1. Compatability Mode SAIL

Mr. Russell Taylor of Stanford University visited BBN during the week of December 11-15, 1972, to help explain aspects of the SAIL system and to contribute to making certain design decisions for TENEX SAIL, which

Stanford was interested in monitoring rather closely.

During Mr. Taylor's visit we constructed a SAIL compiler at Stanford from BBN using the ARPANET. We then created relocatable SAIL files. Upon determining that various loaders at BBN and Stanford were incompatible with these files, a compatible loader was located at Carnegie-Mellon and was imported to BBN over the ARPANET. The relocatable files were then loaded to make a complete SAIL system on BBN's TENEX, using TENEX's 10/50 compatibility mode. Mr. Taylor's test program for SAIL (the "monkey-banana" problem) was then compiled and run on BBN's TENEX.

A problem with the TENEX compatibility mode was fixed permitting local loading to work properly. All required SAIL source files and subsystems from Stanford were then brought to BBN via the ARPANET and a new SAIL compiler was constructed at BBN. The compiler was tested using Mr. Taylor's monkey-banana problem. At this point, versions of SAIL could be created in-house at BBN.

Mr. Taylor gave additional explanations on important parts of SAIL that were not readily comprehensible from the documentation. Finally, we made progress in determining sensible ways to use available TENEX mechanisms to implement SAIL semantics

efficiently and in designing TENEX SAIL.

## B.2. Establishment of TENEX/10-50 Master Files

Early in 1973, we spent three months working at Stanford University on the conversion effort. Trips were also made to JPL to discuss various features of the conversion as well as methods of using the ARPA network to provide for multiple interacting processes. A main goal during this time was to create one set of master source files for the SAIL language, from which 10-50 SAIL or TENEX SAIL could be assembled. It was decided that this set would be maintained at Stanford.

To accomplish the goal of creating one set of master sources, it was decided that switches in the assembly code would be used wherever possible to facilitate specifying options for different operating systems. This approach has several advantages. It helps to ensure that the TENEX-SAIL implementation will compile and run programs written for Stanford SAIL. SAIL development may proceed at Stanford, and TENEX-SAIL will in general benefit from that effort without the tedium of comparing and merging sources after every new release. The effort merely to maintain TENEX-SAIL is reduced to a negligible level. Yet it is possible to develop TENEX-SAIL independently as far as

may be desired to take advantage of the greater power of the TENEX operating system and the capabilities of the ARPANET.

### B.3. A New IO Package

IOSER, the source file for the SAIL IO service routines, was rewritten to take advantage of the TENEX IO system. IOSER (mostly written by Mr. R. Smith of the Stanford University Institute for Mathematical Studies in the Social Sciences) allows the user familiar with 10-50 to continue to use the procedures and calling sequences that are familiar to him, and it allows SAIL programs written at Stanford to compile and run on TENEX.

A set of IO calls oriented toward TENEX is included in IOSER. These calls are easy for the TENEX user to assimilate, and are quite efficient, because they closely follow the organization of the TENEX JSYS's for doing IO.

Because an advanced user of the language is likely to want to mix TENEX and 10-50 IO in one program, the revised IOSER permits him to call the appropriate IO primitive, or if necessary, to escape to machine-language and execute the desired JSYS's, even

though the rest of the program uses "10-50 style" IO. In fact, he can mix the two sets of IO functions at will, with the following exception: the procedure for recognizing the name of a file or device and then initializing it for IO must be completed in one mode or the other. One cannot recognize the file name with the TENEX GTJFN call and then proceed to open it with the 10-50 OPEN. But once the file is ready for the first byte transfer, the user is free to call any IO function, whether it originated as a 10-50 call or a TENEX call.

#### B.4. Additions and Changes

We have revised SAIL's design to tailor its interactive facilities to TENEX and to take advantage of the TENEX virtual memory organization. As examples of the former, the compiler command scanner and the interactive debugger were rewritten to provide such TENEX features as interactive file name recognition and command completion. Reorganization of memory is a more significant area. The location of the runtime support package which SAIL user programs map into their address space to complement the compiler output (called "the Segment") has been changed so that the space available for user data has been considerably increased.

### B.5. Testing and Benchmarks

We have done an extensive amount of testing by running several "benchmark" programs which are known to work in DECUS SAIL and at Stanford. We have successfully run a variety of programs: PTRAN and RTRAN, part of the compiler bootstrap; MONKEY, a theorem-proving demonstration program which uses coroutines extensively; IFN, a program for editing out unwanted assembly switches from FAIL (assembly language) source files, which makes heavy use of IOSER, the largest block of TENEX-specific code in the Segment; and most important of all, TEST, a program from Stanford for checking out a SAIL system.

### B.6. LEAP

One issue that we have been concerned with is how to make LEAP (an associative data base facility embedded in SAIL) use the TENEX features effectively. It looked relatively simple to make the Stanford LEAP work on TENEX, but this LEAP was designed for a swapping monitor, not a more sophisticated demand-paged monitor such as TENEX. An important consideration is that the number of LEAP items is limited in Stanford LEAP to 4K by a 12-bit address field in the basic pointer representation. This assumes a core

restriction which does not exist in TENEX, and 4K is entirely too few items for realistic applications such as robot "world model" data bases. To raise this limit requires changing the internal pointer representation, which is so fundamental and low-level that it implies writing a whole new LEAP. The SAIL group at Case Western Reserve University has recognized this need and has embarked on a large project to write a TENEX LEAP that will provide a very large data base kept on disk, using TENEX file/fork page mapping to bring data into the user's address space as needed. This will result in a LEAP which can be used to solve problems involving very large data bases, yet the user with a small LEAP data base will not be penalized. Case is planning their LEAP around the Stanford version of SAIL, but they have expressed interest in using TENEX SAIL. We believe this LEAP will adequately serve the needs of the JPL robot project, which requires a large data base and reasonable retrieval times.

### C. Graphics System

As mentioned in Section II.D, we have developed a sophisticated graphic display system which dynamically displays the robot in its simulated world. Demonstrations of this system were given to JPL personnel during their visits to BBN. As a result of their interest, we examined hardware/software trade-offs, developed a list of hardware desirable for JPL or essential to running BBN's Time-Sharing IMLAC Monitor (TSIM), and explored the problem of assembling a special version of TSIM. As a result of the study JPL subsequently obtained an IMLAC computer system that would run with TSIM -- thus saving a considerable amount of software development time.

JPL personnel were given a detailed explanation of the capabilities and internal design of the monitor and of how it communicates with the TENEX system. Thus, when the IMLAC arrived, it not only had fully operational software, but JPL personnel could immediately put it to use to run their application programs.

As the need arose, we consulted (via telephone) on various issues of IMLAC programming and hardware design and we provided our TENEX computing facility for JPL personnel to use in building and debugging their IMLAC

programs.

The TSIM Monitor is only a part of the overall graphics display system used in our robot research. The other items described in Section II.D (and also features such as arc and line-drawing software) are available to JPL and have been transferred as the need arose.

#### D. Consultation on the ARPANET Technology

The ARPA computer network consists of many different types of computers tied together via high-speed data links in such a way that the resources available on any one computer are available to each computer on the network. The effective result is that each network user has at his disposal all the computing power that he is likely to need. We have provided consulting, instruction, and documentation which have resulted in the JPL robot project using the BBN and ISI computer systems through the network. Aside from using the ARPA network as a computer access device, JPL and BBN have used the network to enable cooperative debugging and demonstrations of various programs. An interesting example of this occurred when a modified version of our IMLAC display code for the Mars-like environment (described in Section II above) was created and run at JPL via the ARPA Network. By operating in LINKed mode (the IMLAC's at BBN and at JPL tied together via TENEX and the ARPA Network in such a way that Teletype output appearing on one also appears on the other), we were able to control the movement of the simulated robot by means of a simple program written in LISP and FORTRAN.

This interaction provided a means by which JPL could evaluate our robot simulation to determine if it would fit their needs.

As a side benefit of performing this demonstration, we learned how to operate effectively in the LINKed mode and determined some system improvements that will make such interactions simpler in the future.

### E. Consultation on Artificial Intelligence Research

We have provided assistance on an as-needed basis with various members of the JPL staff. This assistance has varied from discussions of basic problems in Artificial Intelligence research to consultation on detailed TENEX programming problems.

Drs. Weinstein and Levine of JPL visited BBN and were given a demonstration of the BBN SCHOLAR system (for semantic information retrieval and reasoning using linguistic facts). We discussed BBN's interests and capabilities in the areas of robotics and storage and manipulation of knowledge, and discussed plans for collaboration with JPL staff on topics of A.I. research.

Additionally, we consulted on issues relating to the computer simulation of a Mars-like environment and robot. This simulation is to be used by JPL in developing the robot system until robot hardware is available to provide real-world input. As a result of these discussions, we considered the possibility that the simulation under development by BBN (described in Section II of this report) may meet the needs of JPL. It was determined that at least several minor modifications would be necessary to fit the display program into JPL's IMLAC computer and to simulate a

simpler "eye".

Finally, the entire context of the research portion of this contract has been made available to JPL through our progress reports and personal contacts.

Appendix AOur Approach to Research on Robot Intelligence

## 1. Intelligence vs. Sensori-Motor Functions

By a "robot", we are envisioning a machine with various sensory and motor capabilities, which is responsible for the gathering of certain information and for the performance of certain tasks (e.g. the exploration of a hostile environment such as that of Mars), and which operates with only limited human intervention. The precise level of human involvement is not important to our discussion; all that matters is that there is some point at which the human yields responsibility for figuring out precisely how a task is to be performed, and that that problem-solving responsibility is taken over by the robot itself.

Our research is concerned exclusively with this "intelligence" aspect of robot performance, that is, with the ability of the robot to figure out for itself how to perform a given task, how to coordinate several on-going goals at once, how to gather desirable information, and how to draw useful conclusions or summarizations from what it has learned. This level of investigation is actually quite distinct from the

"sensori-motor" problems that are the initial hurdles encountered by builders of robot hardware: problems such as finding object boundaries in a televised view of a scene, or computing the optimal way to move a mechanical arm that has many degrees of freedom. From the point of view of our present research, the functioning of the sensori-motor system is important only insofar as it interacts with the robot's problem-solving processes. In other words, the details of how the sensori-motor system works are not of central importance to our investigation of robot intelligence, so we are able to disregard such questions to an extent that might seem surprising at first.

## 2. A "Scientific" Approach to Robot Intelligence

Modern technology has its foundations in modern scientific understanding; for example, the technology of celestial navigation is based on the science of celestial mechanics. Yet strangely enough, most attempts to create a technology of robot intelligence have proceeded in the absence of any scientific understanding of the foundations of intelligent problem-solving behavior. This way of proceeding may be expeditious in the short run, but it is certain to prove very costly in the long run.

It is as though we were to plan orbits for a satellite on a cut-and-try basis. This might work some of the time, but without a knowledge of the principles of celestial mechanics there would be no way of correcting an orbit that turned out to be faulty, and even if the orbit happened to turn out correctly, there would be no reliable way of modifying it or adapting it to another mission. Exactly the same limitations would apply to a robot whose problem-solving system was developed in a cut-and-try manner to perform only a particular set of tasks: we would not know how to correct the system in case it began to fail in some manner, and even if it happened to succeed we would have no good way of modifying or adapting it to another mission.

For these reasons, we have attempted, as much as possible, to gain a "scientific" understanding of the principles that must underlie the operation of an intelligent robot. These principles, which are more abstract than the design of any particular system, should give us the kind of knowledge we need in order to design particular systems to order, to correct them when they go astray, to extend them to new tasks, and to adapt them to more complicated or wholly different missions.

### 3. Our Robot Simulation

Although the principles of robot intelligence are abstract, we cannot discover them by philosophical "armchair" contemplation. We must observe the behavior of functioning robot systems. Unfortunately, the creation and maintenance of actual hardware robots requires attention to a multitude of details regarding the operation of the sensori-motor system, and as we have said these sensori-motor functions are not directly relevant to the "intelligent" level of behavior that we are interested in investigating. Besides, the problems of hardware robot systems are receiving expert attention in projects at the Jet Propulsion Laboratory, the Stanford Research Institute, and other laboratories. Therefore, for our purposes we have chosen the path of simulating a robot and its spatial environment. Our simulation model is realistic enough to confront us with many of the hardest problems in robot intelligence, while at the same time sparing us from many problems in the design of the robot's sensori-motor system. Furthermore, our simulation is easier to modify to new configurations than the average hardware robot, giving us flexibility in the tasks and environments to which we can apply our robot problem-solving system.

#### 4. Psychological Modeling

Our simulated robot in many ways resembles an animal that is interested in exploring its environment. Many of its tasks are similar to those solved by our cats and dogs around the house: to learn the lay of the land, to determine and recognize the important objects in the environment, to discover the best paths from one desirable place to another, and so on. In designing our robot simulation, we have intentionally striven to preserve this animal-like-ness (which is also human-like-ness), especially at the cognitive levels. The principal reasons for doing so are that (a) animals (including humans) are useful as models in that they embody the forms of intelligence that we seek to understand, and (b) we have no way of knowing whether or not there are any non-animal-like principles of intelligent behavior (as some have suggested), so that in fact animals provide the only instance where intelligence is known to exist and to function well in adapting an organism to its real-world environment.

What this means operationally is simply that we have tried to keep our minds open to analogies between our robot and known intelligent systems (i.e. humans and animals), and we have tried to design the robot so as to foster such analogies. In spite of these

inclinations, our robot problem-solving system is still predominantly "artificial" in its organization and behavior. This is true because (a) we are after all programming a computer, and we naturally tend to do things in ways that are suggested by computational efficiency, and (b) the field of cognitive psychology is actually almost devoid of relevant theories or insightful experimental evidence, especially when it comes to the detailed level of modeling that we require in our computer simulation. Therefore, we might go so far as to suggest that simulation models such as ours may contribute more to the understanding of animate cognitive systems than present-day psychology can contribute to the design of inanimate intelligent systems.

Appendix BThe Visual Occlusion Algorithm

In Section II.B.3 we defined the visual occlusion of environmental Feature Points by a Terrain Object intervening between the Feature Points and the robot. In that section we gave an example of how occlusion affects the robot's vision (Figures 5a and 5b), and toward the end of Section II.B.4 we indicated the importance of occlusion for the robot's cognitive model of the world.

The computer simulation of visual occlusion is conceptually a straightforward problem, but there are numerous technical difficulties in producing a simulation which is fast and efficient. We considered a number of algorithms, and programmed several, before we arrived at the method described below, which was conceived and implemented by Mr. Robert Bobrow at BBN.

Let us recall the situation: the robot can see only "Feature Points", which may be either Point Objects or Terrain Feature Points (selected points from the borders of Terrain Objects). The boundary lines which compose Terrain Objects are not themselves visible, but they are able to occlude Feature Points lying behind them. There is a "dominance" ordering

among the various types of objects as follows: mountains, hills, craters, all Point Objects. Each type of Terrain Object is capable of occluding Feature Points of its own type or of types which it dominates (precedes in the above list). Point Objects, having no boundary lines, occlude nothing.

The basis of our method for computing occlusion follows quite directly from the geometry of the situation. First, all the Feature Points which lie within the visual field are determined. Then, for each of these Feature Points, the line of sight from the robot to the Feature Point is computed. The line of sight is considered to be a complete line, and so are the line segments forming the boundaries of all the Terrain Objects. For each boundary line we compute the intersection of the boundary with the line of sight, and then determine if the intersection point actually lies within the boundary line segment and the line-of-sight segment. If so, and if the boundary line segment is part of a Terrain Object which dominates the Feature Point (according to the definition given above), then the Feature Point is occluded by the boundary line and thus is not actually visible. If the Feature Point is not occluded by the boundaries of any Terrain Objects, then it is actually visible within the visual field.

A moment's thought will show that the algorithm above is highly inefficient. If there are  $M$  Feature Points in the visual field and there are  $N$  boundary segments, then roughly  $(M*N)/2$  intersections must be computed and then checked to see if they lie within the segments involved. In addition, the obvious way of determining whether a Feature Point lies within the angular and radial boundaries of the visual field requires computing one or more inverse trigonometric functions and the Euclidean distance from the robot to the point. A simple calculation of the time involved in the above algorithm for a world of several hundred Feature Points and several hundred boundary line segments shows that this technique is an order of magnitude too slow (at least) for a process that must be repeated hundreds or thousands of times in the simulation of an exploratory trip by the robot.

Fortunately, we have been able to find ways of modifying this basic algorithm to bring it up to a useful speed and efficiency. These modifications include the use of various index tables to restrict the number of Feature Points and boundary lines considered to almost a minimum for the given visual field. The algorithm also avoids the computation of inverse trigonometric functions in determining if a Feature Point lies within the angular bounds of the visual

field. The current algorithm can determine the visible Feature Points within a fully-open visual field in less than one second of CPU time on the TENEX system, and it usually requires only a small fraction of that time for normal purposes (in which the visual field is quite small and fairly empty). The algorithm is written as a FORTRAN program which, together with its data, occupies fewer than 10K words in the TENEX system.

There are two primary index tables. One table permits the algorithm to restrict its attention to a (generally small) subset of the Feature Points in the robot's world, by allowing it to quickly find the points which lie within a rectangle just circumscribing the given visual field. For each of these Feature Points the second table permits the algorithm to quickly isolate those boundary line segments which pass through a (generally) thin strip lying along the line of sight from the robot to the Feature Point. The algorithm then computes the intersection of the line of sight with each of these candidate boundary lines in turn. If it finds a boundary line which occludes the Feature Point it starts looking at the next possible visible Feature Point. If there is no occlusion, the algorithm determines which subregion of the visual field the Feature Point lies in. (At this time any Feature Points which happened to lie within the

circumscribing rectangle around the visual field, but not within the field itself, are rejected.)

The algorithm first determines the maximum and minimum X and Y coordinates of the visual field. These are used to determine which elements of the Feature Point array to look through.

For each of the Feature Points whose visibility is to be determined, the coefficients of the line of sight between the robot and the point are computed. These are then used to determine which of the rectangular regions of the boundary line index the line of sight passes through. Only those boundary line segments lying within a rectangle passed through by the line of sight are ever dealt with. The algorithm scans through the elements of the lists in each of the index boxes, and determines if the boundary line actually does occlude the Feature Point. A Feature Point is considered to be occluded if any boundary line is found which occludes it (it is unnecessary to determine which boundary line lies closest to the robot along the line of sight). Thus the computation terminates for any Feature Point as soon as a single occluding boundary line is found.

When a Feature Point is found which is not occluded, the algorithm must determine which subfield of the visual field it lies in. At the same time a final check is made to see if the Feature Point actually lies within the field at all. This requires determining its position relative to four concentric circular arcs and six radial lines. To check for the position of the Feature Point relative to the circular (distance) boundaries we use the square of the distance to the object, comparing it to the square of the radii of the arcs, avoiding the unnecessary computation of the square-root needed for Euclidean distance.

In order to check the position of the Feature Point relative to the radial boundary lines without computing inverse trigonometric functions and without worrying about the circular ordering properties of angles, we use the inner product of vectors. The inner product of two vectors is positive if the two vectors lie within 90 degrees of one another. Thus, by taking a vector 90 degrees clockwise from the radial vector, the inner-product of this new vector and the vector corresponding to the Feature Point is positive exactly when the point is in the half plane to the right of the radial vector. Since the radial boundaries of subfields can never be more than 120 degrees apart, a Feature Point lies between two radial boundaries

whenever it is to the right (clockwise) from the counterclockwise vector and to the left (counterclockwise) from the clockwise vector. Thus, to place the Feature Point within the correct angular segment of the visual field never takes more than twelve multiplications and six additions, with no divisions.

The algorithm will work independently of the dimensions of the arrays, but there is a tradeoff between the size and the computation time. For small sizes a great deal of time is spent in computing intersections (in fact for a 1 by 1 array, the algorithm reduces to the simple case described originally) and for large array sizes a great deal of time is spent in overhead looking through the indices. The current algorithm is programmed for a 40 by 40 index array, which corresponds to a division of the world into a 40 by 40 grid of 10-foot squares.